# **FORMATO PARA EL DESARROLLO DE COMPONENTE FORMATIVO**

|  |  |
| --- | --- |
| PROGRAMA DE FORMACIÓN | Desarrollo de aplicaciones web en tiempo real con JavaScript y NodeJs |

|  |  |  |  |
| --- | --- | --- | --- |
| COMPETENCIA | 220501096 - Desarrollar la solución de software de acuerdo con el diseño y metodologías de desarrollo | RESULTADOS DE APRENDIZAJE | 220501096-1 Describir Las Herramientas Javascript De Acuerdo A Los Requisitos Técnicos Del  Software A Desarrollar. |

|  |  |
| --- | --- |
| NÚMERO DEL COMPONENTE FORMATIVO | 001 |
| NOMBRE DEL COMPONENTE FORMATIVO | Fundamentos de JavaScript |
| BREVE DESCRIPCIÓN | En este componente se abordarán los conceptos fundamentales y básicos del lenguaje de programación JavaScript en conjunto con tecnologías como HTML5, Css3 y frameworks para el desarrollo de aplicaciones del lado del cliente y servidor. |
| PALABRAS CLAVE | Boostrap, Css3, JavaScript, HTML5, NodeJs. |

|  |  |
| --- | --- |
| ÁREA OCUPACIONAL | 2 - CIENCIAS NATURALES, APLICADAS Y RELACIONADAS |
| IDIOMA | Español |

1. **TABLA DE CONTENIDOS:**

**Introducción**

**1. Fundamentos de JavaScript**

1.1. Sintaxis y variables

1.2. Declaraciones

1.3. Operadores y objetos

**2. Poo y estructuras de control**

2.1. Condicionales y clases

2.2. Intervalos y retardos

2.3. Nombramiento y funcionamiento

2.4. Ciclos

2.4. Bloques de código

2.5. Prototipos

2.6. Funciones y declaración de funciones

**3. Tecnologías para el desarrollo Node.js**

3.1. Frameworks back-end

3.2. Introducción al back-end

3.3. NodeJs

3.4 Express

3.5. Ciclos de vida y beneficios

**4. Características y bibliotecas**

4.1. Conceptos TCP

4.1.1. Frameworks front-end

4.1.2. Introducción al front-end

4.1.3 Html5, css3

4.1.4 TypeScript

4.1.5. Angular2

4.1.6. VueJs

4.1.7. ReactJs

1. **DESARROLLO DE CONTENIDOS:**

**Introducción**

En el desarrollo de este componente formativo se hace necesario hacer un recorrido por algunos elementos a tener en cuenta, para ello se debe observar.

****

**1. Fundamentos de JavaScript**

Para reconocer cuáles son los fundamentos de JavaScript, se parte de lo siguiente.

****

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Así mismo, observa los siguientes elementos que complementan lo abordado para dar inicio del componente formativo. 

* 1. **Sintaxis y variables**

JavaScript es un lenguaje no tipado orientado a prototipos, por lo cual su sintaxis es relativamente sencilla, que por lo general es funcional.

Para poder ejecutar algún algoritmo o funcionalidad en JavaScript será necesario utilizar un documento HTML que se renderice en el navegador tal como se especifica en la figura 1.

**Figura 1**

Documento html con script de JavaScript.



Nota. SENA (2022).

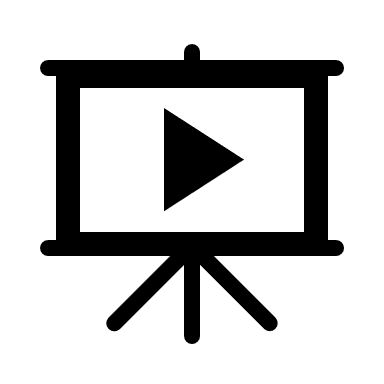
Se debe seguir las siguientes instrucciones:



Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Además, visualiza la información que amplia de la sintaxis y variables



**Variables**

Son identificadores de memoria o contenedores que permiten almacenar valores, para poder hacer uso de estos en el lenguaje de JavaScript primero se deben declarar y asignar un nombre, para posteriormente hacer uso de estos.

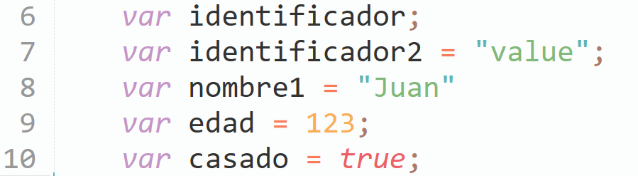
Los identificadores hacen referencia a las constantes o variables que son requeridas para el control de un flujo. Como se mencionó previamente JavaScript es un lenguaje no tipado, por lo cual para realizar una declaración de identificador de memoria este no estará atado a un tipo de dato en específico, tal como sucede en otro lenguaje altamente tipados como Java, C, C++, C#.

Por el contrario, al realizar una declaración solo se establece el espacio de memoria y al realizar la asignación automáticamente ese identificador tomará la especificación del dato asignado.

Por ejemplo, para poder crear un identificador será necesario utilizar la palabra reservada ‘var’ o ‘let’, a continuación se define el nombre del identificador, el cual no puede ser números, no debe tener caracteres especiales y podrá ser una cadena secuencial de caracteres acompañados de números al final, ver figura 6.

Figura 6

Declaración de identificador de memoria



Nota. SENA (2022).

Los datos de una variable pueden estar quemados en el código fuente tal como los expuestos en la figura 6, o se pueden recibir de la interacción con el usuario por medio de elementos de control de entrada, como la figura 7.

**Figura 7**

Ingreso de datos por ventanas emergentes



Nota. SENA (2022).

En JavaScript existen diferentes tipos de datos tal como se enseña a continuación:



Las variables son requeridas dado que con estas se pueden hacer los procesos de programación dinámicos, permitiendo permutar la información a la medida que se ejecutan las funcionalidades implementadas.

* 1. **Declaraciones**

En el desarrollo de software y en el aprendizaje de un lenguaje de programación hay diferentes aspectos a tener en cuenta, entre los cuales está la gramática y sintaxis, para obtener el máximo rendimiento al momento de realizar desarrollos e implementaciones.

**Conceptos básicos**



Entre los conceptos básicos, se puede resaltar que JavaScript es un lenguaje que no tiene distinción entre mayúsculas y minúsculas, por lo cual, si se realiza la definición de una variable en mayúsculas o minúsculas esto hará referencia al mismo identificador de memoria, en otras palabras y dicho de forma técnica es un lenguaje ***case-sensitive***. También es importante recordar que se debe hacer uso de un sistema de caracteres ***Unicode*** por lo que no es correcto utilizar caracteres especiales para definir identificaciones de memoria.

Entre otras de las características básicas de JavaScript, es que al finalizar una línea de código o una sentencia no es necesario poner al final un ‘;’ pero en las reglas definidas por el lenguaje se sugiere que se agregue como una buena práctica.

**Figura 8**

Consideraciones básicas de JavaScript

Texto

Descripción generada automáticamenteUn dibujo de un personaje de caricatura

Descripción generada automáticamente con confianza baja

¡Observa!

Nota.Sena (2022)**.**

En la figura 8, se puede observar que en la línea 7 y 8, al definir el identificador ***primerNombre*** y ***primernombre***, respectivamente, hacer referencia al mismo espacio en memoria, por lo cual, al ser un lenguaje con case-sensitive, inicialmente el identificador tendrá el valor de ‘Juan’ y luego será remplazado por ‘Alex’; ahora en la línea 10 se puede observar que la variable definida como ***cantidadNiños***, es incorrecta dado que utiliza un carácter especial perteneciente al lenguaje español que es la ‘ñ’, esta variable deberá ser renombrada de otra forma y por último se puede observar que las líneas 7, 8 y 10, que se finaliza con ‘;’ recordando que no es una norma del lenguaje, pero es una recomendación de buena práctica.

**Comentarios**

Un comentario es un bloque de código no ejecutable, destinada para dejar anotaciones útiles para para el programador, que por lo general es usado para documentar variables, operaciones, funciones y se pueden aplicar en la mayoría de los elementos del programa. La sintaxis de los bloques de comentarios en JavaScript es igual que muchos lenguajes de programación.

**Figura 9**

Comentarios en JavaScript.



Nota. SENA (2022).

En la figura 9, se puede observar que hay varias formas de crear comentarios, si se requiere un comentario de una sola línea solo con iniciar la sentencia con // será suficiente, pero dado el caso que se requiera que sea un párrafo o más de una línea deberá iniciarse así /\* y cerrarse de la siguiente manera \*/. Adicional a eso recordar que no se pueden anidar comentarios.

Los conceptos básicos del lenguaje abarcan otros temas, los cuales se irán abarcando en el desarrollo de este componente.

* 1. **Operadores y objetos**

Los operadores en JavaScript son signos o símbolos que sirven para definir expresiones, las cuales pueden ser de asignación, lógicas, aritméticas, de comparación y referencia.

**Operadores de asignación**

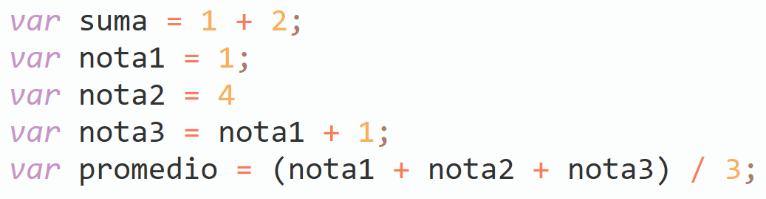
Imagen de la pantalla de un video juego

Descripción generada automáticamente con confianza baja

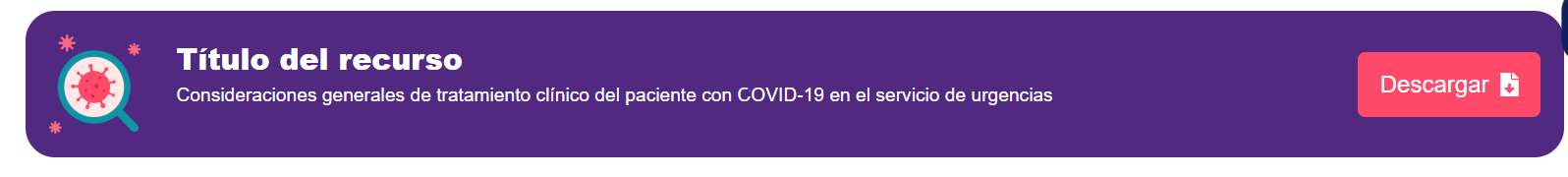
Son aquellos operadores que definen un valor del operando derecho al operando izquierdo, siempre y cuando el operando izquierdo no sea un valor literal, es decir que sea un identificador de memoria, el operando de asignación más utilizado y sencillo de recordar es el igual (=), el cual permite asignar un valor a la variable definida en el lado izquierdo, ver figura 10.

Figura 10

Operador de asignación (=)



Nota. SENA (2022).



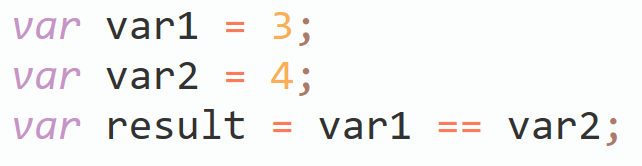
También se pueden utilizar operadores de asignación compuestas tal como la siguiente tabla [Anexo\_ operadores de asignación compuestas](https://docs.google.com/document/d/1YQfRzNXkI6l7sGSRZCgc83TZ5HBw8pO0lVc0htTnys8/edit)

**Operadores de comparación**

Son aquellos que realizan una evaluación entre los operandos y devuelven un valor lógico, es decir un valor de verdadero o falso, los operandos pueden ser números, cadenas, booleanos, caracteres; dado el caso que los operandos no sean del mismo tipo, el intérprete de JavaScript intentará realizar una conversión implícita para hacer la evaluación, ver figura 11.

Figura 11

Operador de comparación (==)



Nota. SENA (2022).

En la figura 11 se puede observar la creación de dos identificadores de memoria ***var1*** y ***var2***, los cuales tienen una asignación de valor respectivamente, posteriormente se crea un identificador denominado ***result***, este almacena el valor de comparar si var1 es igual a var2, el cual es falso; por lo tanto, el valor de ***result*** es el valor de ‘false’, se puede observar así:



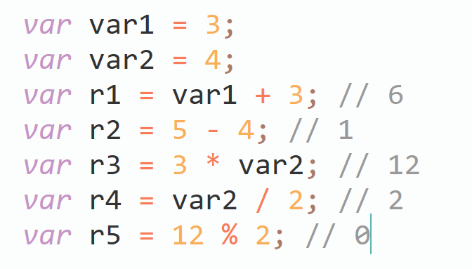
Anteriormente se pudo observar los diferentes tipos de operadores de comparación, es importante recordar que estos ayudan a evaluar dos operandos, retornando con ello un valor booleano de verdadero o falso, esto es útil al construir una sentencia relacional como las que son usadas en los condicionales o algunos de los ciclos.

**Operadores aritméticos**

Son aquellos que realizan operaciones matemáticas, ya sea con valores literales o variables, entre estos operadores encontramos el (+) (-) (/) (\*), ver figura 12.

**Figura 12.**

Operadores aritméticos comunes



Nota. SENA (2022).

Además, se debe observar la siguiente tabla:

|  |  |  |
| --- | --- | --- |
| Operador | Descripción | Uso |
| Residuo (%) | Operador binario o también denominado división modular.  Retorna el resto entero de dividir los dos operandos. | 12 % 5 devuelve 2. |
| Incremento (++) | Suma uno a su operando.  Si se usa como operador prefijo (++x), retorna el valor de su operando después de agregar uno; si se usa como operador sufijo (x++), retorna el valor de su operando antes de agregar uno. | Si x es 3, ++x establece x en 4 y retorna 4, mientras que x++ retorna 3 y, solo entonces, establece x en 4. |
| Decremento (--) | Resta un uno de su operando. El valor de retorno es equivalente al del operador de incremento. | Si x es 3, entonces --x establece x en 2 y retorna 2, mientras que x-- retorna 3 y, solo entonces, establece x en 2. |
| Negación unaria (-) | Retorna la negación de su operando. | Si x es 3, entonces -x retorna -3. |
| Positivo unario (+) | Busca convertir el operando en un número, si este no lo es. | +"3" retorna 3. |
| +true devuelve 1. |
| Operador de exponenciación (\*\*) | Calcula la base a la potencia de exponente, es decir, baseexponente | 2 \*\* 3 returns 8. |
| 10 \*\* -1 returns 0.1. |

Estos operadores solo pueden ser usados en el ámbito matemático, operando valores numéricos, con identificadores de memoria del tipo numérico.

Una caricatura de una persona

Descripción generada automáticamente con confianza baja

**Operadores lógicos**

Son aquellos operadores que se usan para establecer uniones entre valores booleanos, y generar un valor booleano. Tener en cuenta los siguientes ejemplos:



Para completar la información anterior revisar en detalle los datos que se presentan entre operador, uso y la descripción de cada uno de los que se relacionan.

|  |  |  |
| --- | --- | --- |
| Operador | Uso | Descripción |
| AND Lógico (&&) | expresion1 && expresion2 | Tener en cuenta que cada expresión debe retornar un valor lógico true/false. Dado el caso que las dos expresiones sean true, el operador && devolverá true, en caso contrario siempre retornara false. |
| OR lógico (||) | expresion1 || expresion2 | Tener en cuenta que cada expresión debe retornar un valor lógico true/false.  Dado el caso que una de las dos expresiones sean true, el operador || devolverá true, en caso contrario siempre retornara false. |
| NOT lógico (!) | ! expresion | Tener en cuenta que la expresión debe retornar un valor lógico true/false.  Dado el caso que la expresión sea true, al negarse retornara el valor contrario, es decir, false; y en caso contrario ocurrirá lo mismo. |

**Objetos**

JavaScript, es un lenguaje que está construido en un paradigma basado en objetos, recordando que un objeto es una estructura que permite la colección de propiedades, atributos, características o datos, estos a su vez tienen asociado un valor es decir una información relacionada a lo que representa.

Tener en cuenta el siguiente ejemplo:



Estas son algunas de las particularidades de JavaScript para la definición de objetos en el lenguaje.

1. **Poo y estructuras de control**

La POO (programación orientada a objetos) es un paradigma de programación, que pretende solucionar problemas informáticos, realizando la abstracción de las funcionalidades en clases y objetos. Es importante recordar que las clases son una estructura que establece las propiedades y comportamientos, por medio de atributos y métodos, mientras que los objetos son una instancia de lo definido en la clase, dicho de otra manera, la clase es una plantilla y el objeto es una referencia concreta tomada de la plantilla.

¿Cómo se realiza? Observar a continuación:

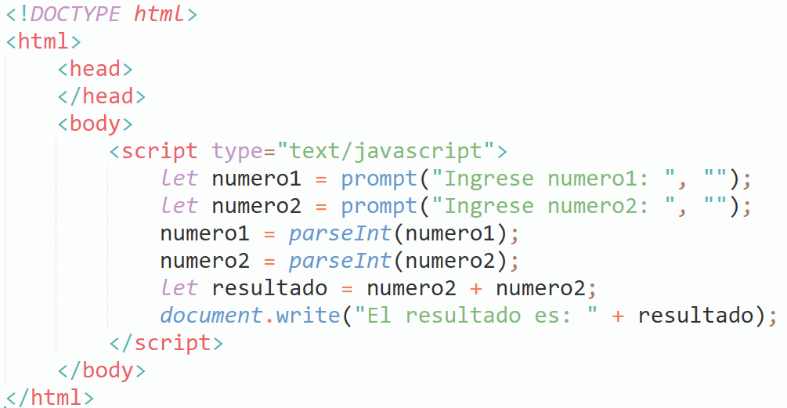


**Estructura secuencial**

Las estructuras de control permiten establecer los flujos para la ejecución de un programa o algoritmo, en el caso de que en un problema o fragmento de código solo exista la entrada, el proceso y la salida se le determina estructura secuencial, ver figura 19.

**Figura 13**

Estructura secuencial



Nota. SENA (2022).

Las soluciones que ejecutan estructuras secuenciales suelen ser los más sencillos, dado que dependiendo de lo que se requiera solucionar se necesitarán de otras estructuras que cambiarán el flujo de control, es de resaltar que no todos los problemas, algoritmos o problemas informáticos se solucionarán con estructuras secuenciales, dado que existirán problemas más complejos, los cuales requerirán tomar decisiones, repeticiones de sentencias con la finalidad de evitar duplicación de funcionalidad, uso de bloques de código para reutilizar funcionalidades, entre otros elementos.

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Observa los elementos de la estructura secuencial



* 1. **Condicionales y clases**

Dado que no todos los problemas se pueden resolver de forma secuencial, se tienen las estructuras condicionales las cuales son las encargadas de poder tomar decisiones, al igual que la vida cotidiana, en los algoritmos informáticos es necesario hacerlo.

Para tomar decisiones en procesos algorítmicos será necesario hacer uso de estructuras condicionales, que de forma resumida permiten tomar un flujo u otro en el código fuente planteado en el programa, es decir a diferencia de la estructura secuencial, en la que todo el código se ejecutaba, de tal manera que de la línea 1 hasta la línea 10 se ejecutaba de forma ordenada y consecutiva, con las condiciones no pasa esto, de acuerdo a las condiciones establecidas en las sentencias de evaluación y los valores dados a las variables de control, existirán bloques de código que no se ejecutarán. Para ello tener en cuenta lo que se comparte a continuación:



Una captura de pantalla de una computadora

Descripción generada automáticamente

Ten en cuenta para las condicionales la siguiente información



**Condicional anidado**

Se debe recordar que cada bloque de código encapsulado en llaves {… } es totalmente independiente a otro bloque de código, por lo cual es posible que se pueda definir una estructura condicional dentro de alguno de los flujos de otro condicional, lo importante en este tipo de casos es que cada condicional realizará la validación de condiciones que no son compatibles, es decir será otro grupo de validación, con otras características, ver figura 14.

**Figura 14**

Condicionales anidados

****

Nota. SENA (2022).

Suponiendo que se tiene una super tienda en la cual venden confitería y prendas de vestir, en el cual tienen apartados para las personas de cualquier tipo de edad, esta tienda quiere saber si quien realiza la compra en su tienda es menor o mayor de edad, dado eso ofrecerá unos beneficios de acuerdo con su edad. Dado que sean menores de edad realizará una bonificación o descuento, si el cliente es menor de edad el beneficio se aplica si se realiza la compra de pastel o helado; por otro lado, si se es mayor de edad, se requiere validar la identidad de género del cliente y con ello aplicar una serie de descuentos de acuerdo con la clasificación.



En la figura 14 se puede observar que hay un condicional que realiza la primera evaluación la cual consiste en saber si el valor de la edad es menor a 18, en el bloque de verdadero se realiza un condicional anidado lo cual evalúa si el usuario seleccionó helado o pastel; por otro lado, en el bloque del sino del condicional principal se realiza la validación de identidad de género, masculino, femenino u otro, lo cual lleva a una serie de descuentos. Si se analiza, los condicionales internos o anidados se pretende evaluar unas condiciones que no son compatibles con la condición de la estructura condicional principal y dada la naturaleza clasificatoria de la misma se requiere de las estructuras internas para realizar una segunda validación póstuma a la clasificación inicial, se puede observar cómo se presenta para estructura switch y las clases.



Por otro lado, la sobreescritura consiste en reescribir un método, por ejemplo, haciendo la abstracción de los seres vivos, hacemos ruido, los perros ladran, los pájaros cantan, los humanos hablan, etc., si tenemos un método en la clase A tal como ***hacerRuido***, y en la clase B un método igual ***hacerRuido***, esto genera una sobreescritura, prevaleciendo el método de la clase más especializada, es decir, la última clase definida.

Para aclarar los puntos dichos previamente analizar y ver la figura 15.

**Figura 15**

Herencia y sobreescritura



Nota. SENA (2022).

De acuerdo con los temas que se han visto hasta el momento se puede analizar que hay varias formas de poder establecer estructuras para generar instancias de objetos, hasta el momento esta es la más nueva, pero a su vez es la más similar a lenguajes altamente tipados.

Interfaz de usuario gráfica, Sitio web

Descripción generada automáticamente

Observa aspectos desde la estructura switch



* 1. **Intervalos y retardos**

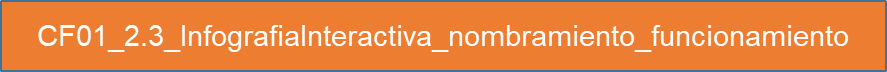
Los intervalos y retardos son bloques de código definidos por el lenguaje para pausar una ejecución o para repetir una instrucción en x cantidad de tiempo, tener en cuenta lo siguiente:



Estas funciones definidas por el core de JavaScript son sumamente útiles, de tal forma que el ***setTimeout*** se usa generalmente para establecer configuraciones, que se deben ejecutar después de cierto tiempo, es decir, si se renderiza en el navegador el html, es pertinente esperar algunos milisegundos para que el DOM pueda identificar todos los objetos creados, ahí se ejecutan las funciones de asignación de listeners entre otros, mientras que los ***setInterval***, se usan para simular procesos en tiempo real, es decir consultar concurrentemente al servidor algún cambio de estado o también para actualizar la información de algún componente en el front.

* 1. **Nombramiento y funcionamiento**

Algunos lenguajes son multiparadigma, es decir que el estilo de programación es funcional con diferentes paradigmas de programación, entre los más comunes es el procedimental y el orientado a objetos, de esta manera se presenta:



Expuesto todo lo anterior, se puede llegar a concluir que POO es un concepto más avanzado para la construcción de aplicaciones, y eso es totalmente correcto, el tema es que muchas de las soluciones en JavaScript son más sencillas que al final emplear siempre POO que complicarían la misma interactividad y el desarrollo de plantillas o páginas webs sencillas. Ahora, si la misión es desarrollar una aplicación web, será necesario utilizar POO, todo dependerá del enfoque que se requiera para el proyecto.

* 1. **Ciclos**

Hasta el momento se han empleado estructuras secuenciales y condicionales, pero, existen otro tipo de estructuras tales como son los ciclos, o las estructuras repetitivas. Estas estructuras permiten ejecutar repetidamente una cantidad de veces necesarias de un bloque de código sin necesidad de escribir de nuevo su código.

Este tipo de estructuras se caracteriza por tener las siguientes características.

* Condición para ingreso al bloque de código.
* Cambio de variable de control para salir del ciclo.
* Sentencias para ejecutarse por cada repetición.



Observa aspectos que se deben considerar en los ciclos



**Ciclo while**

Esta estructura presenta una condición similar al del condicional ***if***, el cual evalúa las condiciones iniciales para dejar ingresar al ciclo, dado que la condición sea verdadera se ingresara al ciclo, en el interior de la estructura se debe garantizar que se realice el cambio de valor de la variable de control del condicional, dado que si este valor nunca cambia y la evaluación de la condición es igual al requerido para ingresar esto generará un ciclo infinito, o como también es conocido un bucle infinito, este tipo de estados es crítico, dado que al dejar un ciclo infinito que consume x cantidad de recursos permitirá que la memoria del programa o en el peor de los casos de la máquina colapsen y lleguen las famosas pantallas azules.

**Figura 16**

While infinito



Nota. SENA (2022).

En la figura 16 se puede observar la palabra reservada ***while***, seguido de paréntesis y una expresión relacional, la cual pregunta si la variable ***contadora*** es menor que 10. Cuando se ejecuta la lógica el valor de la variable es 0 por lo cual esa sentencia devolverá el valor de verdadero, por lo cual entrará al cuerpo del ciclo, en las instrucciones del ciclo se captura un número cada vez que pasa por una iteración y posteriormente se realiza una sumatoria, esta sumatoria se imprimirá apenas se salga del ciclo. Es importante recordar que cuando el flujo llegue la llave que cierra el ciclo él se devolverá a la parte superior del ciclo y volverá a preguntar por la sentencia establecida, en este caso como en el cuerpo del ciclo la variable contadora no cambio, es decir su valor sigue siendo 0, el resultado de esa pregunta será verdadero ingresando nuevamente al ciclo, así de esta manera de forma infinita.

Figura 17

While con iteraciones definidas



Nota. SENA (2022).

En la figura 17 se realiza un cambio en la lógica del código expuesto, el cual consiste en realizar un cambio de valor de variable, de la variable de control de la sentencia condicional del ciclo, por lo cual, el valor de esta variable se modificará de tal manera que cada vez que pase por el ciclo se le añada una unidad, es decir que cuando salga de la primera iteración su valor no será 0, si no 1, y para la siguiente iteración será 2, y así sucesivamente, por lo que en algún momento su valor llegara a 10, en ese caso al realizar la evaluación de preguntar si 10 es menor que 10, el resultado de la sentencia será falso y se saldrá del ciclo, al salir del ciclo la sentencia que está por fuera, imprimirá el acumulado de los números ingresados por el usuario.

**Ciclo do-while**

Esta estructura repetitiva a diferencia que el ***while***, se ejecutará al menos una vez, se debe recordar que, si en el ***while*** la variable de control no cumple con la condición para que la sentencia devuelva un valor lógico verdadero, no ingresará a la estructura, mientras que en la estructura ***do-while***, esto no será impedimento para ingresar, pero si para continuar en el ciclo, esto se debe a que la evaluación en el ***do-while*** se realiza al finalizar la estructura y al inicio tal como en el ***while***, ver figura 18.

**Figura 18**

Ciclo do-while



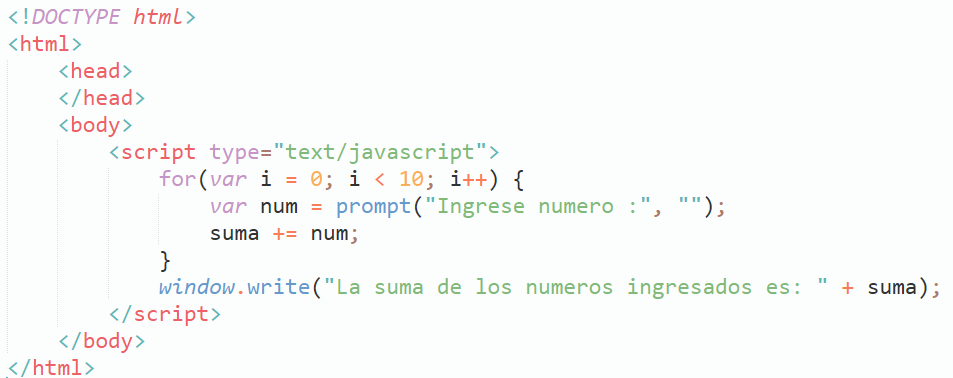
Nota. SENA (2022).

**Ciclo for**

La estructura ***for*** es una estructura más sencilla que regularmente es usada para ejecutar problemas relacionados a ciclos iterativos numéricos, aunque el ***while*** y el ***do-while*** también lo pueden realizar, esta estructura tiene los elementos incorporados para realizar el proceso más sencillo. La estructura del ***for*** contiene en tres subestructuras o argumentos, las cuales son el iniciador de la variable de control, la condición de la variable de control y el incremento/decremento o modificación de la variable de control; su uso es aplicado cuando se conoce el valor de las iteraciones ya sea porque se obtiene de una constante, de una variable o de un valor literal, ver figura 19.

**Figura 19**

Ciclo For, con 10 iteraciones fijas



. Nota. SENA (2022).

Es de recordar que hasta el momento se han determinado diferentes estructuras de control, y es posible que, dependiendo de la necesidad a solucionar, sea necesario que una anide a otra, por decirlo, un condicional anide un ciclo, un ciclo anide a un condicional, un ciclo anide a otro ciclo, etc.

* 1. **Bloques de código**

Los bloques de código son una forma de organizar el código fuente de una aplicación, en JavaScript después de varias décadas de haber salido a la luz, todavía no es muy clara la forma de definir los bloques, aunque a nivel general los desarrolladores después de proyecto insignia, han definido algunos estándares que se siguen a menudo por la comunidad.

Los bloques de código se determinan por las llaves ‘{’ y ‘}’, estos están presentes en declaración literal de objetos, condicionales, ciclos, funciones, entre otros, se podría pensar que estos solo se pueden abrir en ese tipo de estructuras, pero la verdad es que esto no es del todo cierto, ya que un bloque de código se puede abrir en cualquier parte del código.

Esto permite realizar la organización de código, acorde a las necesidades del equipo o del desarrollador como tal (Ver figura 20).

**Figura 20**

Bloques de código



Nota. SENA (2022).

Como se puede observar en la anterior figura, se definen 3 bloques de código, los cuales no son necesarios, ni tampoco interfieren con la lógica funcional de la función test, que se usa para organizar las zonas internas de la función. En JavaScript, hay más de una forma en hacer cada cosa, por ello es importante tener en cuenta cada una y definir cuál de todas será la que se usará en el desarrollo de los

proyectos que llevará a cabo.

* 1. **Prototipos**

Son estructuras que permiten modelar datos que pueden heredar características entre sí similares a las clases en lenguajes altamente tipados como Java, es decir son estructuras que representan la tipología para la agrupación de información, compuesta de atributos, constructores y métodos.

Tal como una función, se puede declarar un prototipo, el cual para posteriormente permitirá utilizar un constructor para establecer una instancia tal como lo muestra la figura 21.

**Figura 21**

Prototipo JavaScript

Texto

Descripción generada automáticamente

Nota. SENA (2022).



Como previamente se comentó, desde el 2015 se incorporaron las clases al lenguaje de JavaScript y con ello características propias de la POO, pero antes de esto se hacía uso de los prototipos que eran la forma como JavaScript podía heredar funcionalidades de otro objeto.

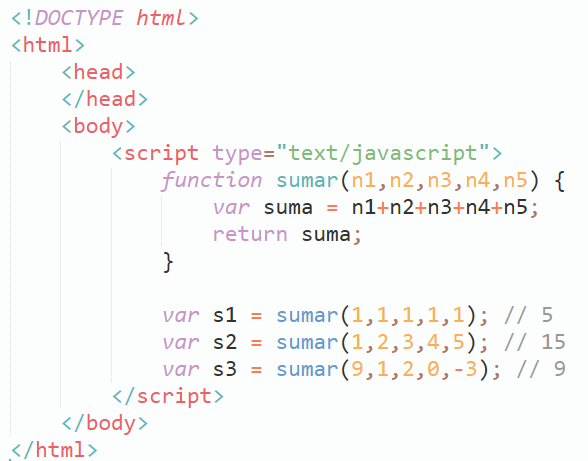
* 1. **Funciones y declaración de funciones**

En la programación de *software* es común tener fragmentos de códigos, que permitan dividir o segregar la lógica. Estos elementos de códigos tener una particularidad funcional o procedimental, de ahí que se nombren procedimientos o funciones. En JavaScript los procedimientos y las funciones, presentan pocas diferencias, dado que es un lenguaje de programación no tipado.

Una función es un conjunto de instrucciones que resuelven una parte del problema y pueden ser llamados desde diferentes partes del programa, es decir, que en varias partes del programa se requiere realizar la sumatoria de 5 números, no sería ideal que por cada vez que se requiera hacer ese proceso se duplique el código de la suma, si no que por el contrario, se define una función que reciba como argumento los valores que se desean operar y devuelve el valor, de esta manera si en la lógica de negocio hay 10 partes en las que se requiere hacer la suma de los 5 números, no será necesario escribir 10 veces el código, si no que se invocará o llamará 5 veces el bloque de código definido como función, esto conlleva una serie de beneficios, tales como dividir los problemas, reutilizar el código, segregar las operaciones, desacoplar las funcionalidades, entre otros, ver figura 22.

**Figura 22**

Función de retorno



Nota. SENA (2022).

En la anterior figura se puede observar, que en la parte inicial del script se define una función de retorno denominada ***sumar***, la cual recibe 5 parámetros presuntamente numéricos y que la función devuelve la suma de esos 5 argumentos. En la parte inferior del script se puede evidenciar 3 llamados a la función ***sumar***, pasando en cada llamado valores dientes, por lo que el retorno en cada llamado será diferente.

1. **Tecnologías para el desarrollo Node.js**

NodeJs es una tecnología de desarrollo de *software*, con la que se puede construir cualquier tipo de necesidad de ámbito general, está basada en JavaScript y es totalmente particular dado que es una solución que está sacada totalmente del ámbito del navegador. Los creadores exaltaron las cualidades del lenguaje y lo abstrajeron a otro contexto haciendo uso del motor v8.

Con NodeJs se pueden construir diversas aplicaciones, actualmente hay varios CRM desarrollados en esta tecnología, aunque se ha vuelto muy popular en el desarrollo de APIs tipo Rest.

Es una tecnología multiplataforma que puede correr en cualquier sistema operativo, ya sea Microsoft, Linux o Mac, para el desarrollo de una aplicación es sumamente sencillo solo es cuestión de instalar el npm y tener un IDE o un editor de código de preferencia y comenzar a codificar.

****

* 1. **Frameworks back-end**

En el ámbito del desarrollo de *software* y especialmente en el desarrollo *backend* hay diversos lenguajes de programación y a su vez por cada uno de ellos hay diferentes *frameworks* de desarrollo. Un *framework backend* es el que se ejecuta del lado del servidor y determina una serie de paquetes, librerías y funcionalidades estándar para el desarrollo en este caso generalmente web.

Los *frameworks* sirven para agilizar el proceso de desarrollo, teniendo en cuenta que ya se cuenta con varios elementos preconstruidos y se pueden extender y reutilizar, permitiendo organizar de mejor manera el estilo de desarrollo y enfocando la mayoría de los esfuerzos en la lógica de negocio y no en configuración o tareas técnicas.

Los más usados son:



Para definir qué lenguaje y *framework* usar, es ideal qué tipo de proyecto y con base a qué arquitectura se delineará, dado que no es igual una aplicación monolítica a una orientada a microservicios, aunque tampoco será igual, implementar un componente en Java donde se necesitaran más recursos de ejecución que en NodeJs que presenta un estilo más liviano para los servidores.

**3.2. Introducción al back-end**

Es muy común dar bastante importancia a los aspectos de las aplicaciones web que son visibles, como lo son las imágenes o el diseño, pero también existen otros elementos muy importantes para que una aplicación funcione de forma correcta y que no son visibles a simple vista.

El ***backend*** es esa parte invisible pero esencial en todo proyecto web, y a la que es de extrema importancia darle toda la trascendencia del caso, porque de él depende en gran parte los procesos informáticos que realiza un sistema.

En el desarrollo web, el *backend*, es la parte del sistema que se encarga de todos los procesos necesarios para que la aplicación web funcione de forma correcta. Estos procesos o funciones no son visibles, algunas de estas acciones que controla el *backend* son la conexión con la base de datos o la comunicación con el servidor de y los procesos o algoritmos que debe realizar la aplicación. A continuación, se presenta como desarrollar un *backend* sencillo que permita interactuar con la base de datos sin prestar muche énfasis en el desarrollo de algoritmos informáticos. Esto debido a que en los procos de pruebas es importante comprender como funcionan internamente las aplicaciones web.

**3.3. NodeJs**

Es un entorno de ejecución multiplataforma de código abierto, fundamentalmente diseñado para el desarrollo del lado del servidor con el lenguaje JavaScript, sus características es que es asíncrono, una arquitectura orientada a eventos y basado en el motor V8 de Google. Se debe recordar que las soluciones desarrolladas en este entorno no se ejecutan en el navegador si no en el servidor web.

**Figura 23**

Desarrollo del lado servidor con NodeJs



Nota SENA (2022).

La particularidad de NodeJs es que incorpora una serie de módulos básicos para dar solución a problemas comunes para los desarrolladores, tales como componentes para conexión a bases de datos, componentes para crear servicios web, para manipulación de archivos, creación de imágenes dinámicas entre otros.

Una ventaja de conocer este *framework* es que habrá homogeneidad en el desarrollo, teniendo en cuenta que tanto el *front-end* como el *back-end* manejarán el mismo lenguaje de programación, y estructuras similares, a pesar de que las tecnologías sean diferentes y estén totalmente separadas.

**3.4 Express**

Es el *framework* más popular de NodeJs para la creación de componentes y funcionalidades del lado del servidor, este marco de trabajo presenta diferentes módulos y componentes, que facilitan el desarrollo.

Se pueden encontrar módulos para:

* Gestión de rutas.
* Renderización de templetes.
* Manejo de sesiones.
* Seguridad, entre otros.

Se caracteriza por simple pero altamente potente, con capacidad de crear aplicaciones robustas y con un alto performance, especialmente tratándose de funcionalidades asíncronas y de tiempo real.

**3.5. Ciclos de vida y beneficios**

NodeJs se basa en el tiempo de ejecución del motor v8 de Chrome, el cual está diseñado en C++, periódicamente surgen actualizaciones, parches de seguridad, rendimiento y compatibilidad con la tecnología base de JavaScript, esto se realiza regularmente para todos los sistemas operativos. El uso de JavaScript en el frontend y backend promueve el uso de Json como transferencia de datos, haciendo que sea una comunicación bilateral más rápida.

La arquitectura de un subproceso permite controlar múltiples conexiones de manera simultánea de manera eficiente discrepando de otras tecnologías que en su caso crearían un nuevo hilo de ejecución, utilizando más memoria RAM en el proceso. Node al operar sobre un solo hilo, hace uso de bucles de eventos y devoluciones para operaciones del tipo E/S, delegando tareas tipo bloqueantes para que se resuelvan lo antes posible tales como las conexiones a las bases de datos.

Cuenta con el módulo de Cluster, el cual permite el equilibrio de carga en varios núcleos de la CPU, adoptando mejor escalabilidad.

Entre las ventajas podemos encontrar:

* Es confiable crear aplicaciones en tiempo real con mucho tráfico.
* Utilizar un solo lenguaje tanto en el servidor como en el cliente.
* Aumenta la eficiencia en el proceso de desarrollo y se minimiza la discrepancia entre el desarrollador front y back.
* El NPM está en constante crecimiento agregando módulos y funcionalidades a usar.
* Se ejecuta más rápido que cualquier implementación en otros entornos.
* Es ideal para el desarrollo de microservicios.

**4. Características y bibliotecas**

Como se ha mencionado a lo largo del desarrollo de este componente, el desarrollo web está compuesto de diferentes tecnologías y componentes, aunque principalmente se debe destacar la tecnología que permite interactuar (frontend) y lo que permite que todo este centralizado (backend).

Cuando se habla de aplicación del lado cliente no solo se habla de la interfaz gráfica de los sitios o aplicaciones web, si no por el contrario, también de las aplicaciones de escritorio, aplicaciones móviles, aplicaciones de Smart tvs, entre otros elementos. Es por medio de estas aplicaciones que es posible interactuar, enviar comandos, indicar acciones, visualizar información, registrar información entre otros, es decir, es la capa más externa, orientada hacia el usuario y es aquí donde se puede evidenciar la experiencia de usuario.

El desarrollo de aplicaciones del lado del cliente específicamente soluciones nativas se pueden encontrar diversas tecnologías, para Android se tiene Android Studio, el cual es una serie de SDK’ donde el lenguaje principal es Java o Kotlin; mientras que para iOS se tiene xcode que al igual que Android estudio tiene sus librerías, teniendo como lenguaje de desarrollo Swift y previamente Objective C; cuando se vira al desarrollo de aplicaciones de escritorio, cada casa puede tener sus propias herramientas de desarrollo, tal como se ve en el mundo mobile, en el caso de Microsoft se cuenta con Visual Studio, y como lenguajes de programación C#, C++, entre otros; Mac como se mencionó previamente hace uso de xcode y Swift; haciendo un paréntesis en este punto, es de recordar que las aplicaciones desarrolladas para Windows no son compatibles con las aplicaciones de OSX, al igual que las desarrolladas para Android no son compatibles con iOS.

También se puede encontrar desarrollos multiplataforma, como es el caso de Java, el cual es un lenguaje interpretado y para cada sistema operativo, tiene su traductor, que en este caso es la JVM. Para el caso mobile, también existen diferentes *frameworks*, aunque estos por lo general se basan en visores web, es decir, el desarrollo es totalmente web, pero con funcionalidades nativas, tales como el caso de Xamarin, Apache cordoba, ionic2, React Native, entre otros.

Específicamente en el desarrollo web, el desarrollo *frontend* solo tiene una serie de tecnologías, las cuales son html, css y javascript, estas son las que son interpretadas y renderizadas por los motores de los navegadores. El *html* cumple la función estructural, por medio de su lista de etiquetas, permite establecer un documento electrónico; el css cumple la función relacionada al aspecto, permite ordenar, dar forma, dar color, tamaño, entre otras características, es por decirlo así la obra blanca de un sitio web; mientras que JavaScript otorga la capa funcional y lógica, todo lo relacionado al control del documento, la algoritmia y funcionalidad es gracias al lenguaje de programación. Se debe recordar que cada uno de esos documentos de forma independiente en el ámbito front no aporta valor al desarrollo front, es decir, solo el html no establece un sitio web, solo el css no establece un sitio web y solo el JavaScript no establece un sitio web, por lo menos deben existir dos de los componentes para poder crear algo y una de las tecnologías fundamentales en este caso es el html, entonces se puede decir que para crear algo, debe obligatoriamente existir la estructura html, pero podría faltar el css o el js.

Desde el inicio de la tendencia de desarrollo de aplicaciones single page, aproximadamente en el 2010, con el surgimiento de Android y a su vez de *frameworks* como apache cordoba, jquery mobile entre otros, comenzó la carrera por desarrollar un *framework* que permitiera sentir la misma experiencia de desarrollar de forma nativa, para la web, en este caso para el front, teniendo en cuenta que al mismo tiempo iniciaba la tendencia de desarrollar aplicaciones cliente-servidor, la cual consiste en separar todo el nivel de acceso, con el nivel de negocio, y separando las tecnologías.

Surgen *frameworks* como Backbone.js, Sencha.js, Angular.Js, Bootstrap css, y a su vez más y más componentes jQuery para optimizar el proceso de desarrollo. Cada uno de estos componentes era sencillo y fácil de implementar, pero a su vez no ofrecía al usuario una forma estándar de implementar las aplicaciones, otro aspecto es que era tedioso el tener que agregar en cada vista los componentes externos para incluir las funcionalidades. Esto fue evolucionando y a la medida que empresas como Facebook, Twitter, Google, Microsoft con sus redes sociales entre otros productos, comenzaron a requerir funcionalidades más complejas iniciaron a promover otros frameworks y formas de trabajo para la web. En el cual surgieron Angular2, React Js, Vue Js, entre otros, siendo estos los más populares en la actualidad.

Imagen que contiene reloj, tabla, esquiando, cubierto

Descripción generada automáticamente

Estos nuevos *frameworks* se caracterizan por presentar una arquitectura MVC o MVVM, con un cantidad considerable de módulos y funcionalidades predeterminadas, módulos de conexión, seguridad, reactividad, entre otros; también es de notar que estos *frameworks* son compilados, dado que por ejemplo en el caso de Angular, está diseñado con TypeScript y para poder ser ejecutando en el navegador debe pasar toda su funcionalidad a JavaScript, permitiendo también de esta forma validar desde la compilación errores de codificación o lógica.

**4.1. Conceptos TCP**

Protocolo de control de transmisión por sus siglas en inglés, es un protocolo de red que permite que entre servidores haya un intercambio de información, permite que los datos lleguen en el mismo orden en el que fueron enviados para que no haya distorsión de la misma.

El rol de este protocolo es controlar que los paquetes de datos lleguen de forma ordenada, confiable y sin errores, también tiene el control de concurrencia, por lo que validara que las peticiones iniciales sean pequeñas y a la medida de la necesidad vayan aumentando poco a poco, tomando como base el ancho de banda de los servidores y las redes que puede soportar.

**4.1.1. Frameworks front-end**

Se comienza por explorar la siguiente información:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Los *frameworks* son una lista de librerías, archivos, funcionalidades, complementos, entre otros elementos que permiten el diseño y desarrollo de sitios web rápidamente. Los *frameworks* implementan funcionalidades que son genéricas en el mundo del desarrollo de *software*, en el frontend, existen para la architectura funcional como los de JavaScript y para temas visuales tales como el de CSS.

Esta es una lista de *frameworks* del lado cliente:

Framework de la capa web.

|  |  |  |
| --- | --- | --- |
| **Tipo de Frametwork** | **Nombre** | **Autor** |
| Framework de CSS | Bootstrap | Twitter |
| Material Design | google |
| awsm.css | Amazon |
| tailwindcss | Tailwindcss |
| Frametwork de JS | Angular | google |
| React.js | Facebook, |
| Vue.js | Evan You |
| JQuery | JQuery |

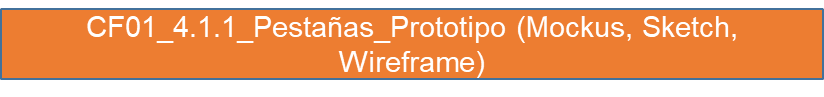
Nota. SENA (2022).

Es relevante que antes de iniciar el proceso de desarrollo de un sistema de información, analizar qué herramientas, componentes y frameworks serán usados para su implementación, esto es importante dado que ayuda a refinar los requisitos funcionales de tal manera que se aprovechen al máximo las capacidades de las herramientas a usar.

**Prototipo (Mockus, Sketch, Wireframe)**

Un prototipo es la elaboración de un producto con los componentes básicos, muchas veces solo es la representación visual de los **elementos** que se desarrollaran en el producto. Esta representación puede ser con técnicas de bajo nivel tales como el *sketch*, que son ilustraciones a mano alzada, más especializadas tipo *mockups* con herramientas de software que permiten construir vistas de forma genérica, tipo wireframe, donde toma mucha más relevancia el aspecto desde la perspectiva del diseño o funcionales con maquetas en html.

Teniendo en cuenta lo anterior, crear un proyecto tipo *mockup* en la siguiente herramienta: <https://balsamiq.cloud/>



**4.1.2. Introducción al *front-end***

El *front-end* hace parte del desarrollo web, en el cual se construye todos los componentes que están de cara al usuario, es decir todo lo visual y relacionado a la interacción de usuario, en otras palabras, lo que también se conoce como el diseño web. Esto tiene en cuenta el contenido, la diagramación, colores, tipografía, elementos gráficos de control, tamaños de letra, color de letra, flujos, navegación, experiencia, usabilidad, entre otros.

Para la construcción de un sitio web, se requieren de diferentes elementos y tecnologías, y hablando solamente del *front-end*, es requerido identificar que no se puede tener un componente sin vincular las tecnologías de html, css y javascript, las cuales en conjunto determinan el front.

El html determina la estructura, las bases del diseño web; el css otorga forma, contextura, agrega valores a las propiedades de los elementos, es decir, es la tecnología que permite dar un look and feel al sitio web; mientras que JavaScript, es la tecnología que permite dar funcionalidad al sitio; si no existiera alguno de los 3 en la construcción de un sitio web, sería casi imposible su óptimo desarrollo.

**4.1.3 Html5, css3**

|  |  |
| --- | --- |
| **Html5** | **CSS3** |
| Es la última versión del estándar de HTML, que se caracteriza por definir las etiquetas de los documentos de la web, es usada para establecer la estructura fundamental de un documento web. Este documento compuesto de etiquetas se valida en el navegador y se renderiza para luego pintar de forma visual los elementos definidos en el código fuente. | En la última versión del estándar de CSS, la cual se caracteriza por permitir dar aspecto y forma a la estructura definida en el html, tales como tamaño, márgenes, bordes, color, fondos, entre otros, en esta versión se añaden de forma nativa animaciones y transiciones que mejoran la experiencia visual del usuario. Es una tecnología tipo script, la cual por sí sola no puede afectar a nada, debe estar acompañada o debe ir de la mano de un documento html. |

**4.1.4 TypeScript**

*TypeScript* es un super conjunto de *JavaScript*, el cual compila en *JavaScript*, es decir su código fuente se escribe en Ts y luego se convierte en Js para ser interpretado. Esto se realiza automáticamente dependiente de la herramienta de pre-compilación ya sea en el *frontend* o el *backend*. Se debe recordar que el navegador web solo puede interpretar archivos JavaScript (ver figura 28).

**Ventajas:**

* Facilidad al implementar aplicaciones complejas.
* Agrega tipos, como String, numbers, booleans.
* Sintaxis clara para definir clases y propiedades, permitiendo el desarrollo orientado a objetos.
* Uso de herencia e implementación de interfaces.
* Captura de errores en proceso de compilación.

**Figura 28**

Estructura tipo clase de TypeScript

****

Nota. SENA (2022).

Este lenguaje es diseñado por Microsoft y es de distribución libre.

**4.1.5. Angular2**

Es un *framework* para el desarrollo de aplicaciones web del lado del cliente, que está desarrollado en TypeScript, es de código abierto y el propietario es Google. Es uno de los *frameworks* más conocidos y populares en la actualidad, al estar desarrollado en TypeScript, permite hacer uso y aplicación de conceptos de programación orientada a objetos.

**4.1.6. VueJs**

Es un *framework* para el desarrollo de aplicaciones web del lado del cliente, directamente desarrollado en JavaScript, cuenta con una arquitectura declarativa y por composición de componentes. El *framework* está centrado en la construcción de la interacción de la vista, enrutamiento, construcción, gestión de estados, entre otras.

**4.1.7. ReactJs**

Es un *framework* para el desarrollo de aplicaciones web del lado del cliente, al igual que Angular Y VueJs, este está desarrollado directamente en JavaScript y es mantenido por Facebook, es de licencia libre, busca implementar una arquitectura MVC o MVVM.

**Síntesis**

El JavaScript es un lenguaje de programación muy utilizado en aplicaciones móviles, al igual que para el desarrollo de páginas web interactivas, pues permite añadirle múltiples funciones; este lenguaje abarca diferentes tipos de software lo que lo ha convertido en el lenguaje de programación más popular, por lo que se hace necesario conocer todo su funcionamiento, características, herramientas, entre otros. Por lo anterior el presente componente formativo abordo todas las temáticas necesarias para conocer a profundidad sobre este lenguaje en conjunto con otras tecnologías con la finalidad de desarrollar diversas aplicaciones necesarias para el mercado. Lo visto en el componente se esquematiza en el siguiente mapa conceptual:

Diagrama

Descripción generada automáticamente

1. **ACTIVIDADES DIDÁCTICAS (OPCIONALES SI SON SUGERIDAS)**

|  |  |
| --- | --- |
| DESCRIPCIÓN DE ACTIVIDAD DIDÁCTICA | |
| Nombre de la Actividad | Maquetación sitio web |
| Objetivo de la actividad | Emplear las tecnologías HTML, CSS y JavaScript para su uso en la maquetación evidenciándose en la plantilla web. |
| Tipo de actividad sugerida | Taller  Evaluación a través de lista de chequeo |
| Archivo de la actividad  (Anexo donde se describe la actividad propuesta) | [Anexo1\_TallerMaquetacion](https://docs.google.com/document/d/1AI5tH0QLFKp-xNfT69g8hkHIMYDSyQPw/edit) |

1. **MATERIAL COMPLEMENTARIO:**

Relacionar el material de apoyo o complementario de los temas abordados en este recurso.

|  |  |  |  |
| --- | --- | --- | --- |
| Tema | Referencia APA del Material | Tipo de material  (Video, capítulo de libro, artículo, otro) | Enlace del Recurso o  Archivo del documento o material |
| Tecnologías para el desarrollo Node.js | Gabriel Coding. (2020). *COMO INSTALAR NODE JS EN WINDOWS 10*. [Vídeo]. YouTube. <https://www.youtube.com/watch?v=BgtB31gXkoA> | Vídeo de Youtube | <https://www.youtube.com/watch?v=BgtB31gXkoA> |
| Poo y estructuras de control | Ortega, D. J. O., & Pérez, M. A. B. (2019). *Bootstrap y Laravel, herramientas para el desarrollo de aplicaciones web*. | Sitio web - PDF | <http://repositorio.upsin.edu.mx/Fragmentos/tesinas/142016030030OvandoOrtegaDenzelJavier10843.pdf> |

1. **GLOSARIO:**

|  |  |
| --- | --- |
| TÉRMINO | SIGNIFICADO |
| Backend: | desarrollo que se realiza del lado del servidor, aplicaciones que centralizan principalmente la lógica del negocio, las bases de datos y los recursos de un sistema para que sean consumidos por una aplicación tipo cliente, o un frontend. |
| Framework: | librería o marco de trabajo que contiene, clases, funciones, configuraciones, paquetes y componentes, que son comunes en el desarrollo de funcionalidades. |
| Frontend: | desarrollo que se realiza del lado del cliente, aplicaciones web, aplicaciones de escritorio, aplicaciones móviles; estas se despliegan en la máquina del usuario. Esta es la capa o el nivel con lo que los usuarios interactúan. |
| JavaScript: | lenguaje de programación tipo scripting que sirve para agregar funcionalidad a documentos web. |
| Localhost: | en el contexto de las redes de computadoras, localhost es una computadora que ejecuta un programa. La computadora que se tiene al frente funciona también como un servidor virtual. Este modo es comúnmente utilizado para realizar pruebas en el desarrollo web. |
| TypeScript: | lenguaje de programación creado por Microsoft, que da supercapas a JavaScript, se caracteriza por añadir características de lenguaje orientado a objetos. |

1. **REFERENCIAS BIBLIOGRÁFICAS:**

Red Hat. (2021). *El concepto de IDE*. <https://www.redhat.com/es/topics/middleware/what-is-ide>

Colaboradores de los proyectos Wikimedia. (2020). Framework para aplicaciones web -Wikipedia, la enciclopedia libre. <https://es.wikipedia.org/w/index.php?title=Framework_para_aplicaciones_web&oldid=128980356>

Fundéu RAE. (2012). *Edición y maquetación de documentos web*. https://www.fundeu.es/escribireninternet/edicion-y-maquetacion-de-documentos-web

NodeJS.org. (2021). *Acerca de Node.js*. Node.js. <https://nodejs.org/es/about/>

npm. (2021). *npm About*. <https://www.npmjs.com/about>

Mozilla. (2021b, junio 9). *Introducción a Express/Node - Aprende sobre desarrollo web | MDN*. MDN Web Docs. <https://developer.mozilla.org/es/docs/Learn/Server-side/Express_Nodejs/Introduction>

Gomez, D. (2017). *Nodemon. vortexbird*. <https://vortexbird.com/nodemon/#:%7E:text=Nodemon%20es%20una%20utilidad%20que,desarrollo%20de%20aplicaciones%20en%20nodojs>

Choque, J. C. (2016). *Adición de estándares de codificación nodejs (1ff84b6a). ·* <https://gitlab.agetic.gob.bo/dbarra/bolivia-libre/commit/1ff84b6ae70f0e9436f6e6ddc7f3e060bc9556ad>

Whyte, A. (2021). *Google JavaScript Style Guide*. Google JavaScript Style Guide. <https://google.github.io/styleguide/javascriptguide.xml>

Express. (2021). *Direccionamiento básico de Express*. <https://expressjs.com/es/starter/basic-routing.html>

Mozilla. (2021b). *await - JavaScript | MDN*. MDN Web Docs. <https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Operators/await>

López, A. (2021). *Qué es Postman y para qué sirve*. OpenWebinars.net. <https://openwebinars.net/blog/que-es-postman/>

chrome web store. (2021). *Postman*. <https://chrome.google.com/webstore/detail/postman/fhbjgbiflinjbdggehcddcbncdddomop?hl=es-419>

Lázaro, G. (2015). Efecto Ripple de Material Design [HTML+CSS+JS]. <https://gersonlazaro.com/tutorial-efecto-ripple-de-material-design-html-css-js>

Material Design. (2021). The color system. <https://material.io/design/color/the-color-system.html#color-usage-and-palettes>

1. **CONTROL DEL DOCUMENTO**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Nombre | Cargo | Dependencia  *(Para el SENA indicar Regional y Centro de Formación)* | Fecha |
| Autor (es) | Gustavo Adolfo Rodríguez Quinayas | Experto Temático | Regional Cauca, Centro de Teleinformática y Producción Industrial | Junio de 2022 |
| Leydy Jhuliana Jaramillo Mejía | Diseñador instruccional | Regional Distrito Capital- Centro de Gestión Industrial | Junio de 2022 |
| Rafael Neftalí Lizcano Reyes | Asesor pedagógico | Regional Santander - Centro Industrial del Diseño y la Manufactura. | Junio de 2022 |
| Carolina Coca Salazar | Asesor pedagógico | Regional Distrito Capital- Centro de Diseño y Metrología | Junio de 2022 |
|  | Jhon Jairo Rodríguez Pérez | Corrector de estilo | Regional Distrito Capital - Centro de Diseño y Metrología | Agosto de 2022 |

1. **CONTROL DE CAMBIOS**

**(Diligenciar únicamente si realiza ajustes a la Unidad Temática)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Nombre | Cargo | Dependencia | Fecha | Razón del Cambio |
| Autor (es) |  |  |  |  |  |