**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 | 220501096-02- Construir aplicaciones web en tiempo real de acuerdo con los requerimientos establecidos. |

|  |  |
| --- | --- |
| NÚMERO DEL COMPONENTE FORMATIVO | 002 |
| NOMBRE DEL COMPONENTE FORMATIVO | Aplicaciones web en tiempo real con JavaScript |
| BREVE DESCRIPCIÓN | En este componente, se abordarán los conceptos fundamentales y básicos del lenguaje de programación JavaScript, en conjunto con tecnologías como comandos básicos, Node.js, servidor socket y requerimientos de *hardware* y *software* de IDE, para el desarrollo de aplicaciones del lado del cliente y servidor. |
| PALABRAS CLAVE | JavaScript, HTML5, Css3, bootstrap, NodeJs |

|  |  |
| --- | --- |
| ÁREA OCUPACIONAL | 2 - Ciencias naturales, aplicadas y relacionadas |
| IDIOMA | Español |

1. **TABLA DE CONTENIDOS**

**Introducción**

**1. Preparación del entorno**

1.1. Requerimientos *hardware* y *software* de IDE

1.2. Configuración e instalación de entorno NodeJs

1.3. Comandos básicos

**2. Codificación web.**

2.1. Creación proyecto Node.js

2.2. Creación e instalación de servidor Express sobre NodeJs

2.3. Configuración de Apache para aplicación cliente

2.4. Codificación de petición tradicional HTTP, síncrona

2.4.1. Codificación de petición tradicional HTTP, asíncrona

2.5. Codificación de servidor socket con NodeJs

2.6. Codificación de cliente socket con JavaScript

2.7. Tabla de comparación entre implementaciones

1. **DESARROLLO DE CONTENIDOS**

**Introducción**

En el uso de las aplicaciones que aportan a la programación de *software*, se inicia este componente formativo, a partir del cual se invita a observar los siguientes elementos como parte de su introducción



**1. Preparación del entorno**

NodeJs llegó para quedarse y, como lo expresan los desarrolladores del ***frontend,*** es un beneficio para todos aquellos que trabajan con JavaScript, ya que, para codificar en el ***backend,*** entre otros, no será necesario aprender otro lenguaje. En este momento, es posible crear aplicaciones bidireccionales, en las que tanto el cliente como el servidor pueden comunicarse a fondo en tiempo real e intercambiar datos, de hecho, ***NodeJs*** ha promovido y revolucionado los desarrollos implementando funcionalidades en tiempo real con ***WebSockets***.

Esta tecnología es empleada por muchas compañías, entre las cuales se pueden encontrar:



Analizando la operación de las compañías mencionadas, es posible destacar su operación en el ámbito del servicio o producto, y sus desarrollos dependen de NodeJs, para poder implementar estas funcionalidades que son críticas, y es que NodeJs se puede aplicar para cualquier solución, de cualquier negocio.

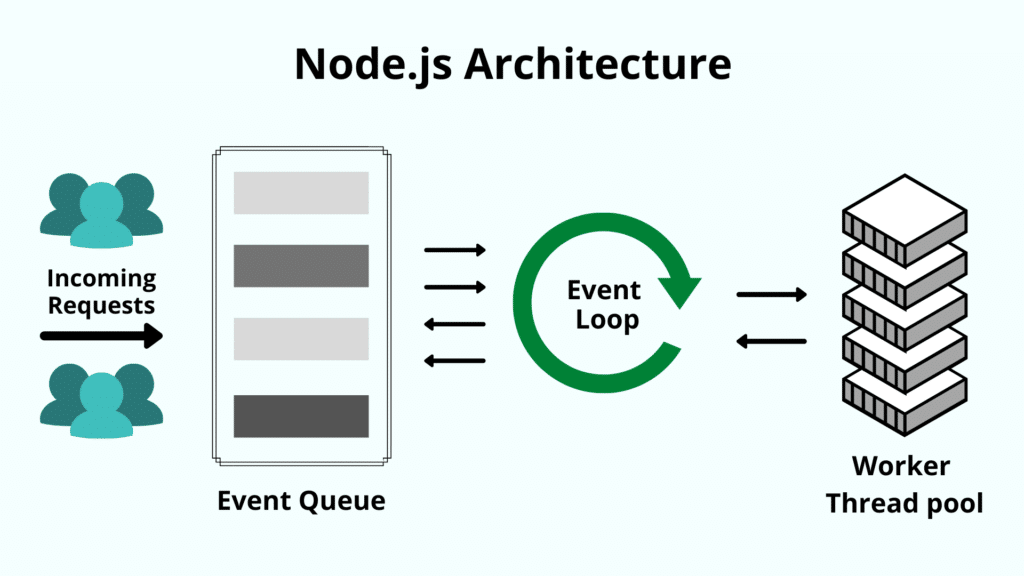
Algunas de las particularidades y ventajas de usar NodeJs son las siguientes:

* Sintaxis simple, heredada de JavaScript.
* Una curva de aprendizaje ágil y sencilla.
* Capacidad de escalar rápidamente cualquier solución.
* Código abierto y flexible.
* Desarrollo de aplicaciones multiplataforma, es compatible con Windows, Linux y OSX.
* Ser fullstack con un solo lenguaje de programación.
* Desarrollos en *real time*.
* Alta comunidad y activa.

NodeJs está fundamentado en una arquitectura ‘Single Threaded Event Loop’, el que permite manejar varios clientes al mismo tiempo. Esto es un modelo solicitud-respuesta multihilo, en el que cada petición se controla por un hilo aparte, con lo que agiliza el proceso de solicitudes concurrentes.

**Figura 1**

*Arquitectura NodeJs*



.

NodeJs tiene una forma particular de funcionar, explicada en la siguiente manera:



Teniendo en cuenta que NodeJs utiliza menos hilos, por lógica, utiliza menos recursos CPU / memoria y ejecuta más rápido las tareas.

**1.1. Requerimientos *hardware* y *software* de IDE**

Para instalar ***NodeJs,*** no se tendrán requerimientos muy complejos; el primero y más importante es conceptual, y es tener un conocimiento previo del lenguaje de programación ***JavaScript***; por otro lado, en segundo lugar, es muy importante tener conocimientos en POO (Programación orientada a objetos).

**Requisitos de sistema**

NodeJs no requiere una configuración compleja para poder instalarse, prácticamente en cualquiera de los equipos usados en esta época se puede ejecutar ***NodeJs***, incluso en tarjetas de Arduino puede ser instalado y ejecutado. Solo se vería afectado si hay otro aplicativo que aparque gran porcentaje de memoria, de lo contrario, a no ser que se tenga un equipo extremadamente viejo, ***NodeJs*** se ejecuta con normalidad.

**IDE o Editores de Texto**



****

Una de las principales diferencias entre un IDE y un editor de texto es que el primero, por lo general, contiene más ayudas, ***plugins*** y componentes que apoyan el proceso de desarrollo, aunque también y más importante, es que, por lo general, contiene un compilador, lo que permite que el código fuente sea chequeado antes de ser empaquetado y desplegado, esto los hace herramientas complejas y pesadas, tanto para descarga como en consumo de recursos, por ejemplo .***Net framework*** y Windows Forms; para poder crear una aplicación en estas tecnologías, será necesario utilizar Visual Studio, que es el IDE de Microsoft para crear aplicaciones para Windows. Por otro lado, los editores son, por decirlo de una manera, una parte de un IDE, donde se pueden codificar aplicaciones, y el proceso de compilación se realiza o delega a otro componente externo. Una particularidad de este tipo de herramientas es que son muy livianas y consumen menos recursos.

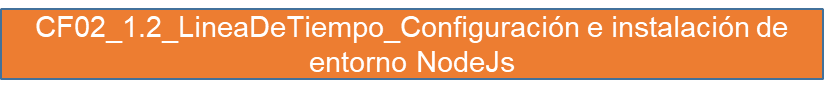
Algunas recomendaciones para usar son las siguientes:



Como se puede concluir, para el desarrollo de aplicaciones web, especialmente de NodeJs, se pueden encontrar diferentes IDE con los cuales construir sistemas sorprendentes, todo dependerá de cómo se use cada herramienta y de la preferencia del equipo de desarrollo o del desarrollador, para inclinarse por uno u otro.

**1.2. Configuración e instalación de entorno NodeJs**

En este punto, se va a recordar el paso a paso para instalar NodeJs. Si ya se realizó previamente, solo valide qué versión tiene en la actualidad; para ello, observe:



**1.3. Comandos básicos**

Después del proceso de instalación, se habrán adquirido varias herramientas que permitirán el desarrollo de aplicaciones con NodeJs, entre las cuales se encuentra el ***npm*** (Node Package Management). Observe a continuación:



Con esto, se puede validar que Node y el npm están correctamente instalados y se puede proseguir a realizar la construcción y configuración de proyectos.

**2. Codificación web**

El proceso de maquetación web es la transformación del diseño o prototipo a un producto web en un conjunto de archivos (html, css, js), de manera que serán desplegados (renderizados) por un navegador.



**Uso de estilos**

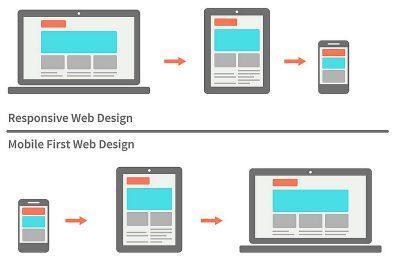
El uso de las hojas de estilo comprende aplicar cambios en las propiedades de los elementos html determinados en la maquetación o definición del diseño, esto es uno de los beneficios determinados por los *frameworks* CSS, que proporcionan una estructura html con la configuración predeterminada de los componentes. Es decir, que si se tiene un elemento tipo tarjeta, se usará, probablemente, el tag <article> o <div>, y se le aplicarán los estilos necesarios, ya sea color, tipografía, tamaño de letra, títulos, párrafos, etc.

***Movie first design vs. Responsive web design***

Algunos de los *frameworks* CSS definían clases, con atributos y estructuras prediseñadas, con la finalidad de tener plantillas que se pudieran redimensionar de acuerdo con la pantalla del dispositivo en el cual se estuviera renderizando, a este tipo de diseños se les denomina adaptativos, es decir que, al minimizar la resolución, se minimiza el tamaño del contenido.

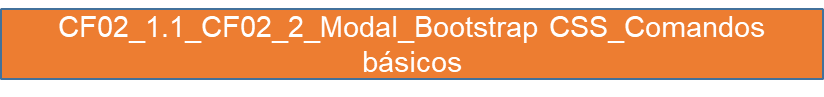
**Figura 2**

*Métodos de diseño*



*Mobile first,* «los móviles primero», consiste en realizar la maquetación teniendo en cuenta los dispositivos móviles, proporcionando la relación de pantalla de los móviles, las *tablets* y las pantallas de ordenador. Estos diseños están pensados para ser escalados de acuerdo con la dimensión de la pantalla en la que son renderizados, es decir, permitiendo cambiar el tamaño y estructura de un tipo a otro.

Algunos de los *frameworks* CSS son:



Bootstrap permite crear interfaces de usuario limpias y adaptables a dispositivos y pantallas de cada tamaño, algunas de sus principales características son:

* Documentación y soporte amplio con HTML5 y CSS3, que permite un uso flexible.
* Un sistema Grillas (o GRID) que permite diseñar usando unas grillas de 12 columnas, donde se pueden establecer contenidos. Esto permite construir contenido *responsive* de forma fácil, intuitiva y rápida.
* Compatible con la mayoría de los navegadores.

***Material Design***

Es un tipo de diseño creado por Google, establecido desde junio de 2014, orientado específicamente para aplicaciones nativas Android desde la versión Lollipop. Se ha publicado una guía (disponible en <https://material.io/design>) para que se realicen aplicaciones web; los ítems propuestos son:

* Uso de lenguaje visual, en conjunto con principios de buen diseño.
* Ser una línea de diseño que permita una experiencia uniforme en diferentes plataformas y dispositivos, tales como pantallas táctiles, control de voz, etc.

Se presentan también los tres principios fundamentales:

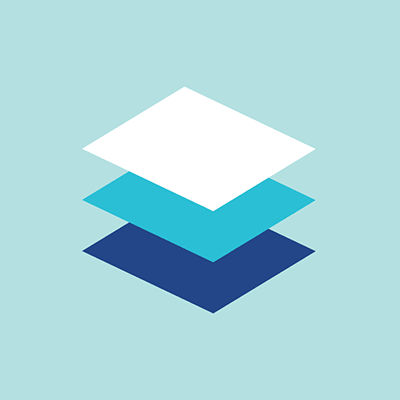
1. **Lo material como una metáfora**

Como lo material está unido a la realidad, fusionándose con el espacio, el tiempo y el movimiento, está ligado a la realidad táctil, es por esto que se puede integrar con la tecnología, haciendo uso de la creatividad y la imaginación.

Los bordes y superficies de las cosas materiales entregan pistas visuales basadas en la realidad. El uso de atributos familiares al mundo real ayuda al usuario a entender las posibilidades de lo que se puede hacer con cada elemento presente en la interfaz. Para lograr esto, se deben presentar los principios superficie, luz y movimiento, como elementos principales para transmitir cómo los objetos existen y están dispuestos a interactuar uno con el otro o con el usuario.

**Figura 3**

*Materiales*

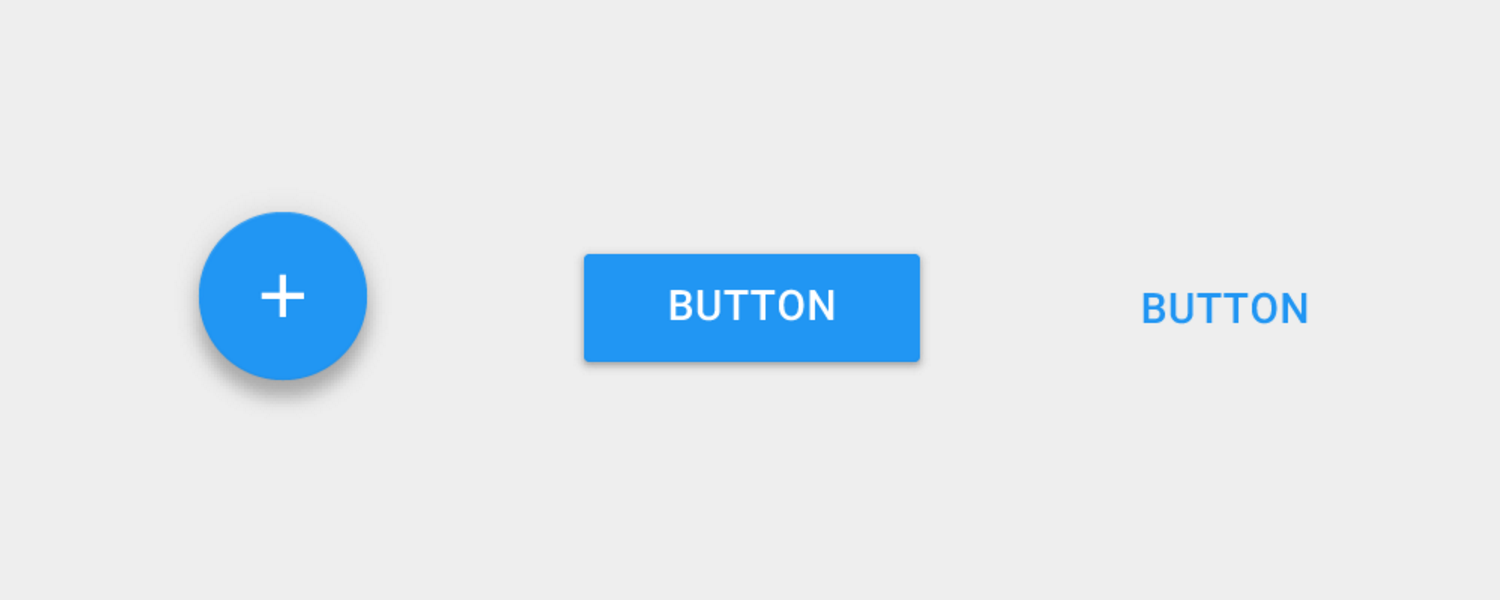


1. **Llamativo e intencional**

Todos los elementos visuales son diseñados por los conceptos de la teoría del diseño, como el uso del color, los espacios en blanco, sombras, el sistema de tabla de datos, etc. Todos estos elementos ayudan a determinar la jerarquía y lectura visual.

**Figura 4**

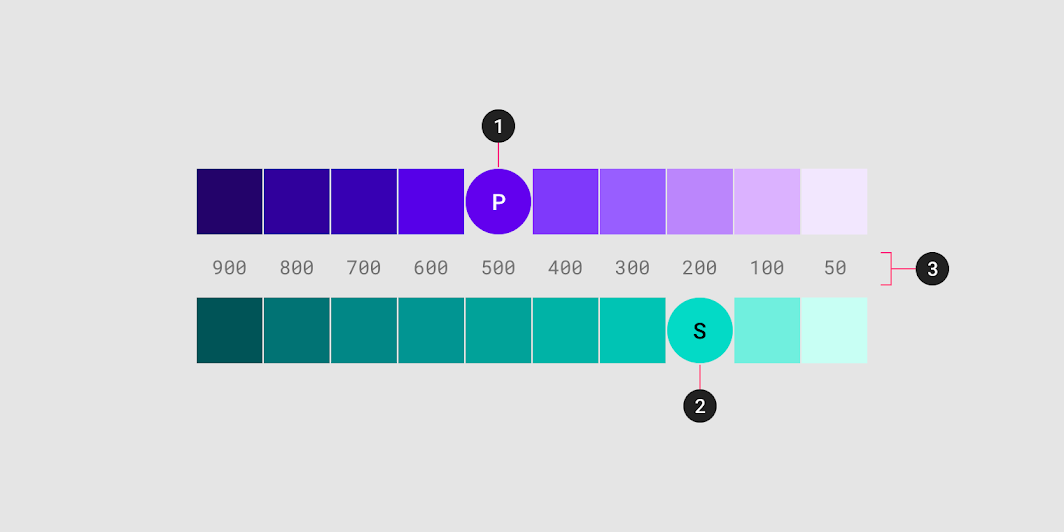
*Diseño de componentes*



El sistema de color de *Material Design* ayuda a aplicar color a las interfaces de usuario de una manera simplificada. En este sistema, el diseñador define un color como primario y otro como secundario, para representar el tema de su marca. Las variaciones de oscuras y claras para cada color se aplican a la interfaz de usuario de diferentes formas de manera preestablecida.

**Figura 5**

*Manejo de colores*



1. **El movimiento otorga significado**

En una aplicación web o móvil, una animación capta la atención de los usuarios, sin embargo, es deseable que estas animaciones no interrumpan la experiencia de usuario. De esta forma, es deseable que exista relación entre las animaciones y la propia esencia del sitio web o aplicación, por ejemplo, el efecto de presionar el botón, la experiencia real de desplazar un elemento.

Para Google, el movimiento enfatiza el papel del usuario como el actor principal de la interacción con la máquina y el que dirige los movimientos en dicha interacción. Las acciones del usuario tienen el poder de cambiar el diseño o disposición de los elementos. Es así que los elementos de la interfaz son presentados al usuario sin interferir con la experiencia de usuario, incluso cuando dichos elementos se transforman y reorganizan.

En conclusión, *Material Design* es una propuesta de diseño novedosa y que se está incorporando en las nuevas versiones de Android. Los principios del *Material Design* van guiados a mejorar la experiencia de usuario teniendo en cuenta las últimas tecnologías. Tenga en cuenta:



Los objetivos que se obtienen con las pruebas de interfaz son:

* Las características que tiene la interfaz deben probarse para asegurar que las reglas del diseño, la estética y el contenido visual estén a disposición del usuario sin lugar a errores.
* Los elementos individuales de la interfaz deben probarse de forma unitaria (por ejemplo, se prueba HTML dinámico, cgi, carrito de compras, etc.) y, con especial cuidado, aquellos que manipulan la presentación de datos y componentes visuales.
* La interfaz debe probarse frente a cada historia de usuario o caso de uso, para descubrir errores semánticos y facilidad de uso por parte del usuario.
* La interfaz debe probarse dentro de una diversidad de ambientes para asegurar su compatibilidad (celular, *tablets*, pantallas, *smart tv*, etc.).



**Pruebas semánticas:** Mientras se prueba cada historia de usuario o caso de uso, se revisa la información en pantalla o *display* que el usuario obtendrá, especialmente, en los menús y formularios, que sea entendible y coherente con el usuario y el contexto en que esté empleado el sistema.



**2.1. Creación proyecto Node.js**

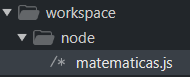
Una de las primeras diferencias entre JavaScript y NodeJs es el concepto de Módulo, el cual contiene variables, funciones, objetos, que son exportados para ser usados en otras implementaciones.



Para entender esto un poco más, creemos el siguiente archivo.

**Figura 6**

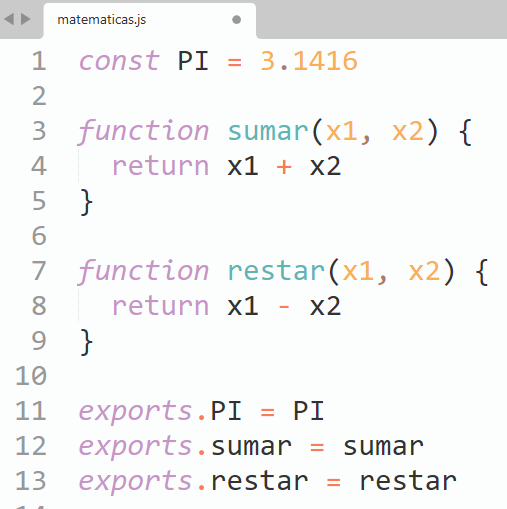
*Creación modulo matemáticas en NodeJs*



A diferencia de los archivos js de JavaScript, en NodeJs no es requerido un documento HTML para ser ejecutado, sino, por el contrario, se usa el intérprete de NodeJs. En el archivo creado, se definirán algunas funciones básicas y constantes.

**Figura 7**

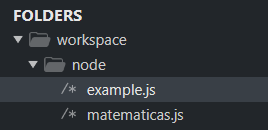
*Definición de funciones y constantes en modulo NodeJs*



En la parte inferior de la definición del módulo, se puede observar que se exportan los elementos, tales como las funciones y las constantes. Luego de definido el módulo, este podrá ser usado en otra sección del código, es decir, todo lo que fue exportado podrá ser reutilizado; si se requiere de alguna función interna del módulo y que no sea visible en otras secciones, solo basta con no exportarlo, pero podrá ser usado al interior del módulo en otra función.

**Figura 8**

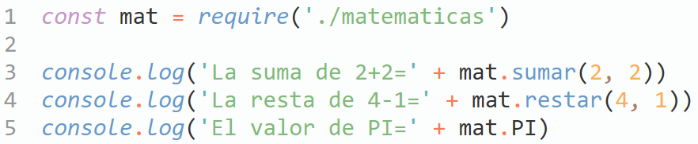
*Creación de archivo para importar módulo*



En el archivo *example.js,* se realiza la importación del módulo construido, llamado *matematicas.js*, en el cual se podrá hacer uso de las funcionalidades exportadas, empleando, de esta manera, el concepto de reutilización de código.

**Figura 9**

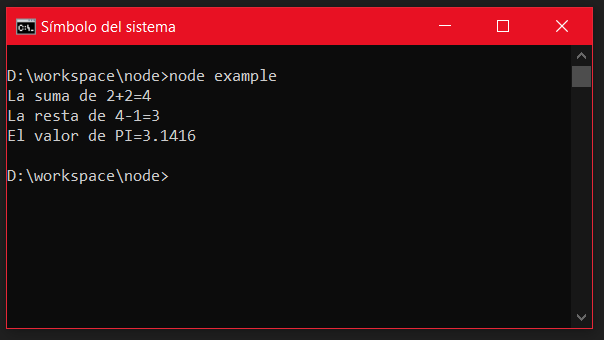
*Implementación de funcionalidad importando módulo*



Para ejecutar el bloque de código de NodeJs, se requiere ingresar a la consola CMD y navegar hasta donde está almacenado el archivo fuente; posteriormente, con el prefijo *node,* buscar el archivo que se quiere implementar.

**Figura 10**

*Ejecución de funcionalidad desde CMD*

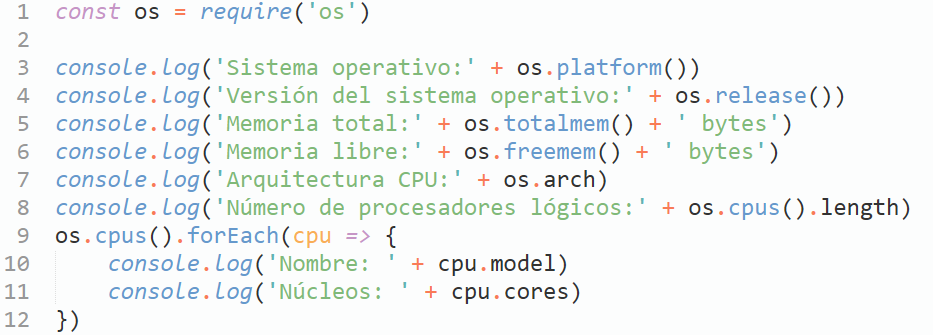


Cabe resaltar que, en la actualidad, hay muchos *frameworks*, con componentes y módulos con funcionalidades genéricas desarrolladas previamente por empresas u otros desarrolladores, lo cual busca optimizar y agilizar el desarrollo de aplicaciones.

También se pueden consumir módulos definidos en el *core* de NodeJs; entre estos módulos, se pueden destacar los siguientes: os, http, fs, net, url, path, dns, process, entre otros más.

**Figura 11**

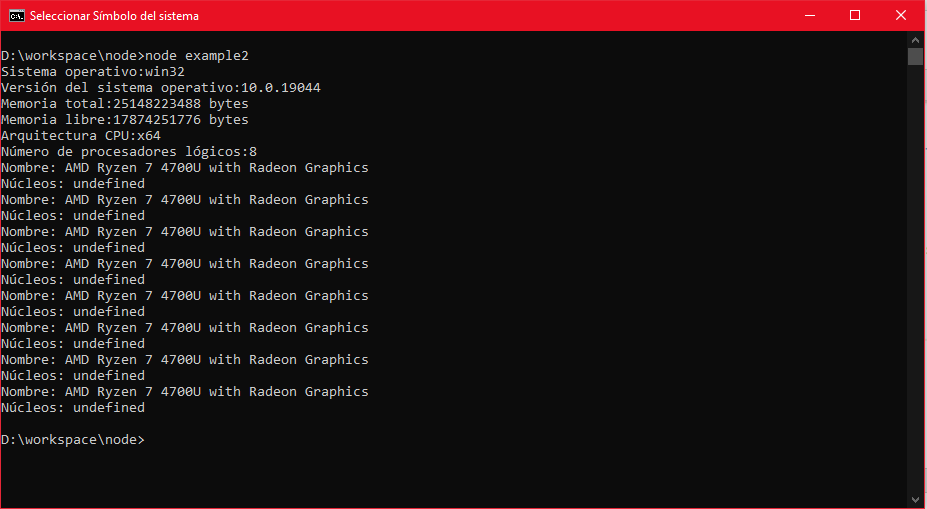
*Consumo de componente del core*



Como se puede apreciar en el anterior ejemplo, se incorpora el módulo ‘os’, este permite obtener información del sistema operativo en el cual está corriendo NodeJs.

**Figura 12**

*Resultado de consumo módulo ‘os’*



Hasta el momento, se han tratado temas relacionados con aplicaciones ejecutadas desde el compilador, apoyadas en la consola CMD. Siga las siguientes instrucciones para construir el primer proyecto tipo servidor con páginas web estáticas. Detalle los pasos que se presentan a continuación.

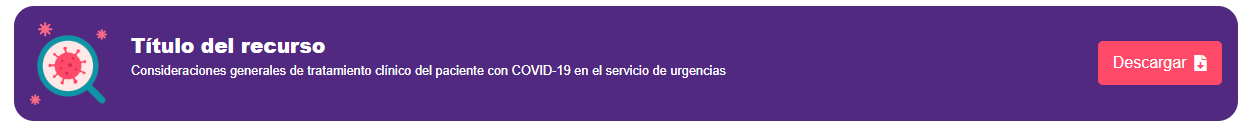


En el anterior ejemplo, se hizo uso de dos módulos del *core* de NodeJs, los cuales son ‘http’ y ‘fs’, permitiendo establecer un servidor web y manejar los archivos html estáticos.



**2.2. Creación e instalación de servidor Express sobre NodeJs**

Con NodeJs, es posible desarrollar un sitio web completo para renderizar páginas web estáticas o dinámicas, aunque la comunidad ha desarrollado un *framework* denominado Express, el cual facilita y ordena el desarrollo de sitios web.



**2.3. Configuración de apache para aplicación cliente**

Tenga en cuenta la siguiente información:



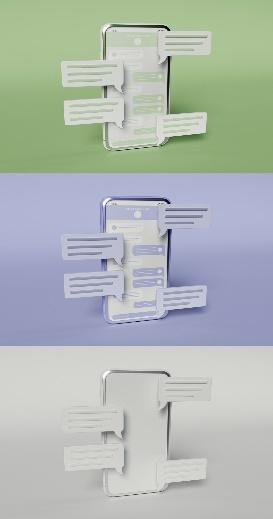
**Instalación de XAMPP**

El proceso de instalación descrito es para máquinas con sistema operativo Windows; para otros sistemas operativos, el proceso puede variar.



**2.4. Codificación de petición tradicional HTTP, síncrona**

El desarrollo de aplicaciones sincrónicas presenta una serie de limitaciones, teniendo en cuenta las capacidades de lo que presenta la red, por lo cual es determinante la aplicación de tecnologías alternativas. Por ejemplo, imagine una aplicación relacionada con el mercado bursátil donde se plasman los valores actuales del mercado al momento de ingresar la aplicación, una aplicación sincrónica mostraría esta información de manera indefinida hasta que una acción de usuario permita realizar la actualización de forma manual.



Lo mismo pasa en un chat. Imagine que el chat de Facebook es sincrónico; no sería posible ver, en tiempo real, las respuestas de las personas con las cuales estamos trabajando, básicamente, se requeriría que actualizásemos el chat para poder ver si realmente ha llegado un nuevo mensaje. Este tipo de interacciones ha sido reemplazado por funcionalidades asincrónicas, que, posiblemente x cantidad de tiempo, permiten una actualización de la información.

***2.4.1. Codificación de petición tradicional HTTP, asíncrona.***

Las aplicaciones asíncronas, por su diseño, permiten notificar cambios al cliente cuando en el servidor han ocurrido cambios, de esta forma, permite que la actualización de información sea más eficiente y rápida. Para la implementación de un flujo asíncrono, es posible encontrar tres formas:

**Figura 61**

*Ejecución de flujo poll*



Nota. Tomado de <https://hvalls.dev/posts/polling-long-polling-push>

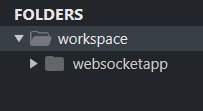
El desarrollo de aplicaciones asíncronas permite una interacción más natural entre el usuario y el producto, permitiendo así una respuesta más eficiente.

**2.5. Codificación de servidor socket con NodeJs**

Para la creación de un servidor socket, será necesario que cree un nuevo espacio de trabajo denominado ***websocketapp*** en la raíz de su *workstation*.

**Figura 13**

*Creación de espacio de proyecto websocket*



Posteriormente crear el archivo de lógica de negocio, denominado index.js.

**Figura** 14

*Creación de archivo de control index.js*



Codificar la lógica del servidor

**Figura 15**

*Código fuente de archivo de control index.js*



Como se puede observar en la anterior imagen, en la línea 7, se hace uso del objeto ***io***, el cual pertenece al módulo de ***socket.io,*** y permite establecer el servidor socket; en la línea 8, establece la conexión de este; y en la línea 9, emite el evento para que los suscritos al socket lo puedan recibir.

**2.6. Codificación de cliente socket con JavaScript**

Es requerido que, para hacer una conexión a un servidor socket, se cree un cliente socket; se requiere que esta implementación se desarrolle del lado del cliente, es decir, será necesario desarrollar una página html y, por medio de esta estructura, hacer uso de las facultades de JavaScript del lado del cliente.

Para ello, será necesario crear un archivo estático en la raíz del proyecto, con el contenido html, y transcribir el siguiente código fuente.

<!DOCTYPE html>

<html>

  <head>

    <title>Socket.IO chat</title>

    <style>

      body { margin: 0; padding-bottom: 3rem; font-family: -apple-system, BlinkMacSystemFont, "Segoe UI", Roboto, Helvetica, Arial, sans-serif; }

      #form { background: rgba(0, 0, 0, 0.15); padding: 0.25rem; position: fixed; bottom: 0; left: 0; right: 0; display: flex; height: 3rem; box-sizing: border-box; backdrop-filter: blur(10px); }

      #input { border: none; padding: 0 1rem; flex-grow: 1; border-radius: 2rem; margin: 0.25rem; }

      #input:focus { outline: none; }

      #form > button { background: #333; border: none; padding: 0 1rem; margin: 0.25rem; border-radius: 3px; outline: none; color: #fff; }

      #messages { list-style-type: none; margin: 0; padding: 0; }

      #messages > li { padding: 0.5rem 1rem; }

      #messages > li:nth-child(odd) { background: #efefef; }

    </style>

  </head>

  <body>

    <ul id="messages"></ul>

    <form id="form" action="">

      <input id="input" autocomplete="off" /><button>Send</button>

    </form>

    <script src="/socket.io/socket.io.js"></script>

    <script>

      var socket = io();

      var messages = document.getElementById('messages');

      var form = document.getElementById('form');

      var input = document.getElementById('input');

      form.addEventListener('submit', function(e) {

        e.preventDefault();

        if (input.value) {

          socket.emit('chat message', input.value);

          input.value = '';

        }

      });

      socket.on('chat message', function(msg) {

        var item = document.createElement('li');

        item.textContent = msg;

        messages.appendChild(item);

        window.scrollTo(0, document.body.scrollHeight);

      });

    </script>

  </body>

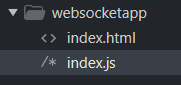
</html>



Este archivo debe estar ubicado en la raíz del proyecto.

**Figura 16**

*Creación de archivo estático index.html*





Se realiza un ajuste en el controlador del proyecto, para renderizar la vista estática. Modifique el index.js con el siguiente código:

const app = require('express')();

const http = require('http').Server(app);

const io = require('socket.io')(http);

const port = process.env.PORT || 3000;

app.get('/', (req, res) => {

  res.sendFile(\_\_dirname + '/index.html');

});

io.on('connection', (socket) => {

  socket.on('chat message', msg => {

    io.emit('chat message', msg);

  });

});

http.listen(port, () => {

  console.log(`Socket.IO server running at http://localhost:${port}/`);

});



Añada los siguientes archivos en la raíz del proyecto.

App.js:

{

  "name": "Chat-SENA",

  "description": "Chat SENA",

  "success\_url": "/",

  "keywords": [

    "node",

    "express",

    "socket.io",

    "realtime",

    "websocket"

  ],

  "scripts": {

  },

  "addons": [

  ],

  "env": {

    "BUILDPACK\_URL": "https://github.com/heroku/heroku-buildpack-nodejs"

  }

}



Package.json

{

  "name": "chat-sena",

  "version": "0.0.1",

  "description": "Primer aplicación con socket.io app",

  "dependencies": {

    "express": "^4.18.1",

    "socket.io": "^4.1.2"

  },

  "scripts": {

    "start": "node index.js"

  }

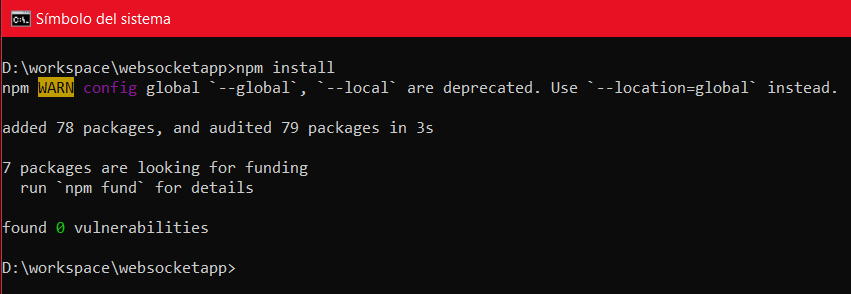
}



Ejecutar el comando > *npm* *install*

**Figura 17**

*Ejecución de npm install del proyecto para configurar dependencias*

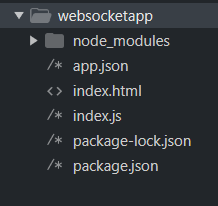




Después de ejecutar el anterior comando del proyecto, debe configurar los módulos definidos en el proyecto, viéndose de la siguiente manera:

**Figura 18**

*Validación de archivos de proyecto*

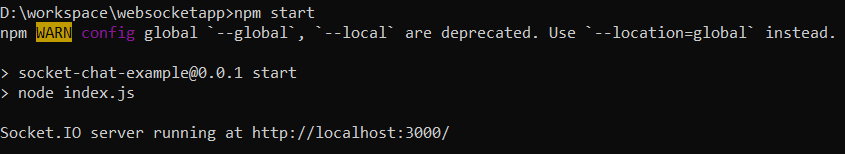




Luego, se requiere ejecutar la aplicación, para probar si el cliente y el servidor socket están funcionando adecuadamente, con lo cual se debe ejecutar el comando > *node index* o *npm start*.

**Figura 19**

*Ejecución de proyecto*

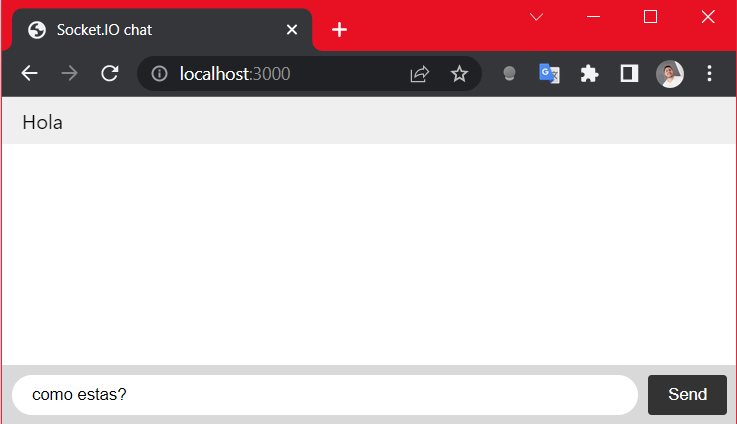




Luego de que se ejecuta el proyecto, se puede proceder a probar la aplicación en el navegador.

**Figura 20**

*Renderización en navegador de vista index*



En la raíz del servidor <http://localhost:3000>, se renderiza la plantilla estática, e internamente se conectará con el servidor socket del *backend*. Como se observa en el pantallazo, cada vez que se dé en el botón *Send*, se enviará un *request* al servidor con la data del mensaje; esta data será devuelta a los clientes que estén suscritos, es decir, al mismo cliente, y posteriormente, renderiza el mensaje en el contenido html.

**Síntesis**

En la actualidad muchas de las aplicaciones web desarrolladas tanto de correo como de mensajería, tienen alto contenido de JavaScript; por esta razón y con el objetivo de crear páginas innovadoras y dinámicas, es necesario que conozca los requerimientos del *hardware* y *software* de IDE, los comandos básicos, la configuración e instalación del entorno, es decir, es necesario tener claros los conceptos de preparación del entorno al igual de la codificación web, por lo que este componente formativo resume a través del siguiente esquema todos los temas abordados y que cumplen la finalidad que se tiene sobre el conocimiento que se debe apropiar para el diseño de este tipo de aplicaciones, esto es:

Diagrama

Descripción generada automáticamente

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

|  |  |
| --- | --- |
| DESCRIPCIÓN DE ACTIVIDAD DIDÁCTICA | |
| Nombre de la Actividad | Maquetación sitio web con bootstrap |
| Objetivo de la actividad | Fortalecer los conocimientos de maquetación empleando las tecnologías de HTML, CSS y JavaScript, que permitan el uso de plantilla web y librerías CSS. |
| Tipo de actividad sugerida | Taller |
| Archivo de la actividad  (Anexo donde se describe la actividad propuesta) | [Anexo1\_TallerMaquetacion](https://docs.google.com/document/d/11bsQ5COGZugVLThKknWL32rb-3r-W6J4/edit) |

1. **MATERIAL COMPLEMENTARIO**

|  |  |  |  |
| --- | --- | --- | --- |
| Tema | Referencia APA del Material | Tipo de material  (Video, capítulo de libro, artículo, otro) | Enlace del Recurso o  Archivo del documento o material |
| Curso de NodeJs | Moisset, D. (s. f.). *Node.js Ya*. Tutoriales Ya. <https://www.tutorialesprogramacionya.com/javascriptya/nodejsya/> | Tutoriales | <https://www.tutorialesprogramacionya.com/javascriptya/nodejsya/> |

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*. |
| Cliente: | son aplicaciones que permiten consumir información de un servidor, toda aplicación de escritorio que tiene comunicación con un servidor es un cliente. |
| *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 el 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. |
| Socket: | proceso asíncrono en el cual dos procesos pueden intercambiar información. |
| Servidor: | son aplicaciones que permiten servir o disponer contenido para el consumo de otras aplicaciones. |

1. **REFERENCIAS BIBLIOGRÁFICAS**

Luna, F., Peña, C. y Iacono, M. (2018). *PROGRAMACIÓN WEB Full Stack 16-Webs dinámicas con AJAX y PHP: Desarrollo frontend y backend-Curso visual y práctico*. RedUsers.

Navarrete, L. y Cárdenas, M. (2016). *Diseño y desarrollo de un CMS (Content Management System) para código abierto utilizando Node.JS y Mongo.DB como base de datos documental*. Universidad Politécnica Salesiana. <http://dspace.ups.edu.ec/handle/123456789/12424>

Paterna, R. (2015). *Desarrollo de una aplicación colaborativa síncrona-asíncrona con persistencia en un entorno web*. Universidad Politécnica de Madrid. <https://oa.upm.es/41062/>

Puciarelli, L. (2020). *Node JS-Vol. 1: Instalación-Arquitectura-node y npm (Vol. 1)*. RedUsers.

Puranik, D., Feiock, D. y Hill, J. (2013). *Real-time monitoring using AJAX and WebSockets*. IEEE.

Trejos, O. (2018). WhatsApp como herramienta de apoyo al proceso de enseñanza y aprendizaje de la programación de computadores. *Educación y ciudad*, (35), p. 149-158. <https://dialnet.unirioja.es/servlet/articulo?codigo=6702430>

1. **CONTROL DEL DOCUMENTO**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Nombre | Cargo | Dependencia | Fecha |
| Autor(es) | Gustavo Adolfo Rodríguez Quinayas | Experto Temático | Regional Cauca - Centro de Teleinformática y Producción Industrial | Julio de 2022 |
| Leydy Jhuliana Jaramillo Mejía | Diseñadora Instruccional | Regional Distrito Capital - Centro de Gestión Industrial | Julio de 2022 |
| Rafael Neftalí Lizcano Reyes | Responsable Equipo de Desarrollo Curricular | Regional Santander - Centro Industrial del Diseño y la Manufactura | Julio de 2022 |
| Darío González | 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) |  |  |  |  |  |