Guía Práctica – Walter Arias Aguirre

DEFINICIONES TÉCNICAS

Node.js es un entorno en tiempo de ejecución multiplataforma, de código abierto, para la capa del servidor (pero no limitándose a ello) basado en el lenguaje de programación JavaScript, asíncrono, con E/S de datos en una arquitectura orientada a eventos y basado en el motor V8 de Google.

V8 es el entorno de ejecución para JavaScript creado para Google Chrome. Es software libre desde 2008, está escrito en C++ y compila el código fuente JavaScript en código de máquina en lugar de interpretarlo en tiempo real.

Node.js funciona con un modelo de evaluación de un único hilo de ejecución, usando entradas y salidas asíncronas las cuales pueden ejecutarse concurrentemente en un número de hasta cientos de miles sin incurrir en costos asociados al cambio de contexto.

Este diseño de compartir un único hilo de ejecución entre todas las solicitudes atiende a necesidades de aplicaciones altamente concurrentes, en el que toda operación que realice entradas y salidas debe tener una función **callback.**

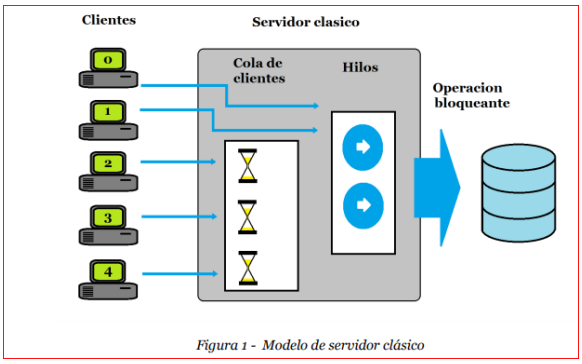


Figura 1 - Modelo de servidor clásico

MODELO DE SERVIDOR BACKEND CLASICO

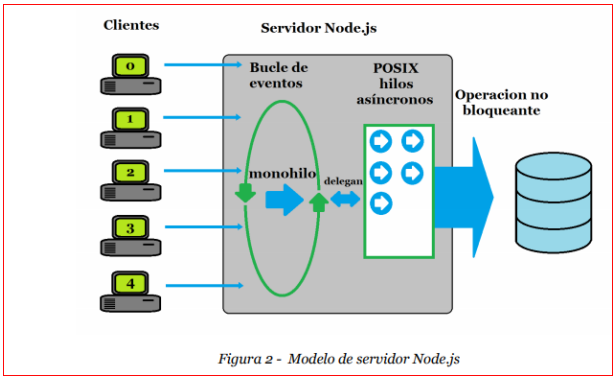


Figura 2 - Modelo de servidor Node.js

MODELO DE SERVIDOR BACKEND NODE JS

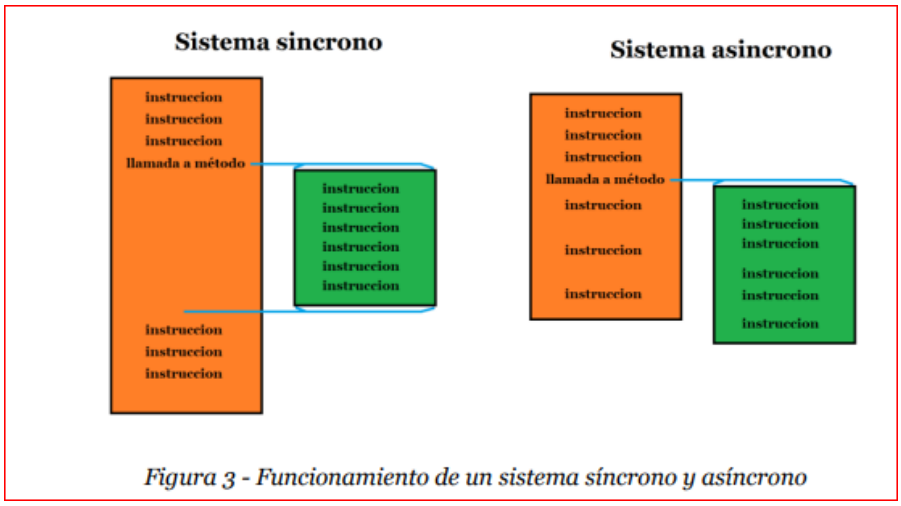


Figura 3 - Funcionamiento de un sistema síncrono y asíncrono

Cuando un sistema síncrono ejecuta una llamada, las instrucciones posteriores a esa llamada no se ejecutan hasta que esta ha sido completada.

Node.js es un sistema asíncrono. Esto significa que las llamadas y métodos son ejecutados de forma secuencial, pero sin esperar a que la anterior llamada haya finalizado.

Cuando un programa está sometido a esperas constantes como lectura de disco, llamadas a métodos de mucho coste, entradas de teclado, etc. Para que el programa se ejecute lo antes posible es necesario reducir esos tiempos de espera. Una solución consiste en continuar la ejecución del código sin esperar a la finalización de la llamada, a esto se le llama ejecución asíncrona.

CASOS DE ÉXITO DE NODE.JS

NETFLIX, PAYPAL, UBER, LINKEDIN, EBAY (SISTEMAS CONCURRENTES, MUCHAS PETICIONES!)

Link recomendado: https://www.toptal.com/nodejs/por-que-demonios-usaria-node-js-un-tutorial-caso-por-caso

TENER EN CUENTA

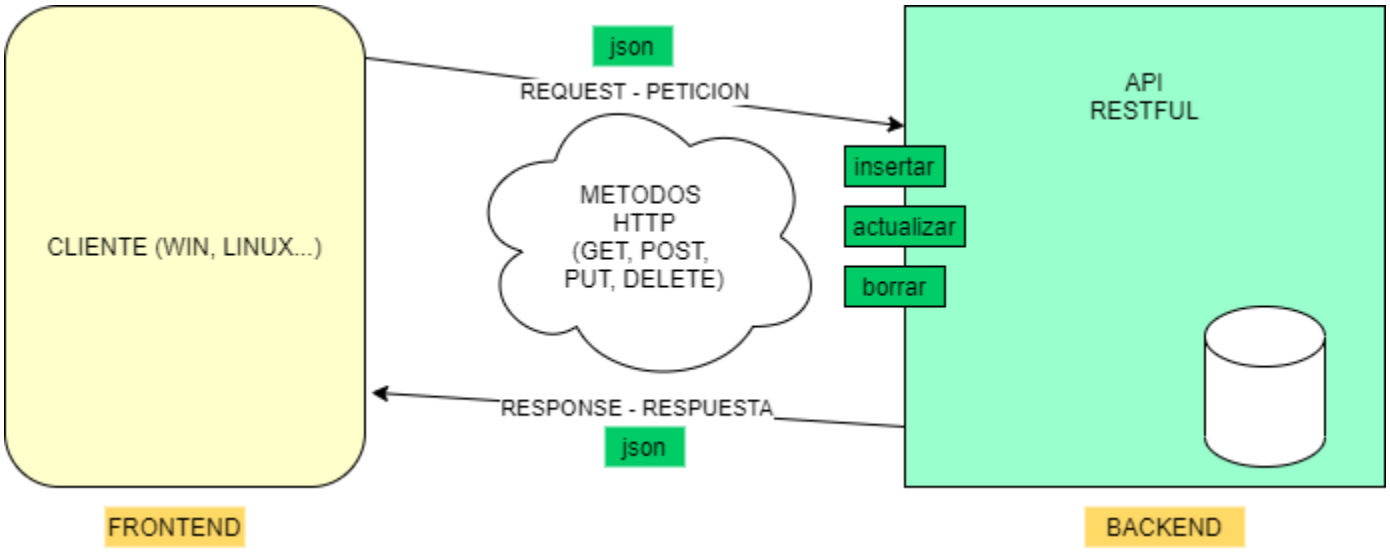
Cuando se trata de cómputo pesado, Node.js no es la mejor plataforma. En general, cualquier operación de uso intensivo de CPU anula todas las ventajas de rendimiento y bloquearía cualquier petición entrante de un subproceso.

Ejemplos de uso intensivo de cálculo (CONSUMO DE CPU): la liquidación de 10.000.000 de facturas del servicio de energía de una ciudad, cálculos de astronomía, simulador de escenarios de economía, simulador de fenómenos de física (Termodinámica, hidráulica, cinemática, Animación CGI)

APLICACIÓN C.R.U.D. DE EJEMPLO

CRUD (Create, Read, Update, Delete) es un acrónimo para las maneras en las que se puede operar sobre información almacenada. Es una abreviatura para las cuatro funciones del almacenamiento persistente. CRUD usualmente se refiere a operaciones llevadas a cabo en una base de datos o un almacén de datos, pero también pude aplicar a funciones de un nivel superior de una aplicación como soft deletes donde la información no es realmente eliminada, sino es marcada como eliminada a través de un estatus

DIAGRAMA DE LA APLICACIÓN



TECNOLOGIA A USAR:

POSTMAN

NODE JS

EXPRESS JS (https://expressjs.com/es/) (LIBRERÍA)

MODULO MYSQL

MODULO CORS

PRIMEROS PASOS:

SCRIPT DE LA TABLA DE EJEMPLO

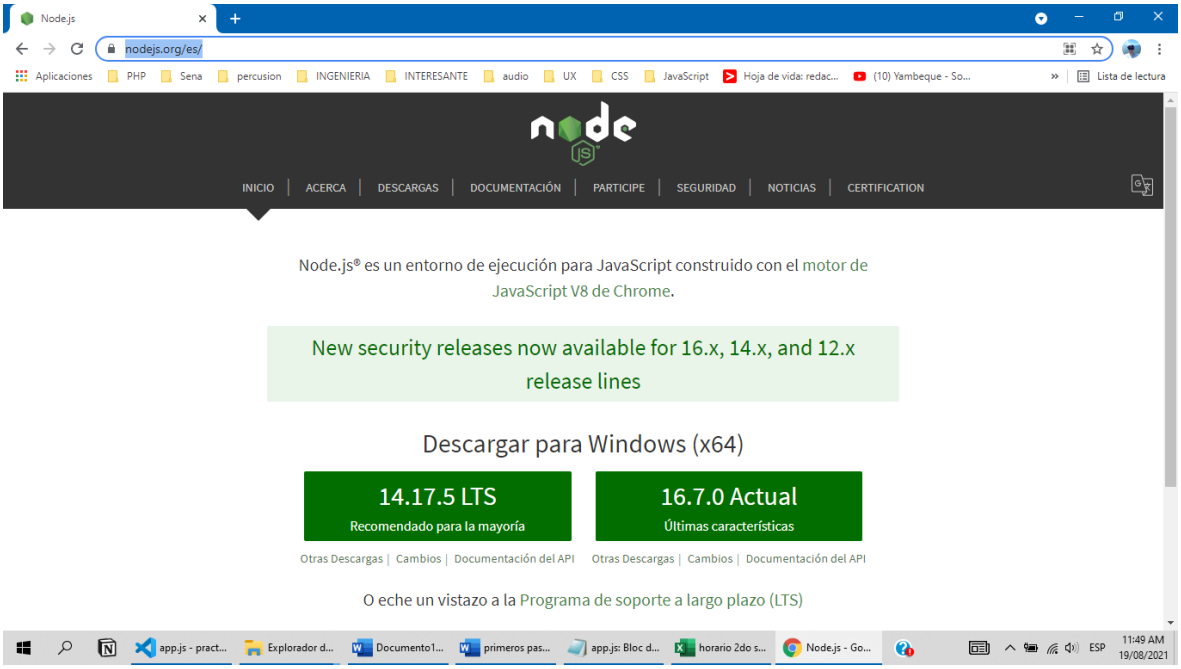
BASE DE DATOS FERRETERIA

```
CREATE TABLE `articulo` (
  `codigo` int(11) NOT NULL,
  `descripcion` varchar(200) NOT NULL,
  `precio` int(11) NOT NULL,
  `existencia` int(11) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

--
-- Volcado de datos para la tabla `articulo`
--

INSERT INTO `articulo` (`codigo`, `descripcion`, `precio`, `existencia`) VALUES
(1, 'Cemento Gris', 25000, 1000),
(2, 'Hierro en chipa', 500000, 1000),
(3, 'Alambre No 12', 5000, 10000),
(4, 'Plafon Bombilla', 3000, 200);

--
-- Índices para tablas volcadas
--


--
-- Indices de la tabla `articulo`
--
ALTER TABLE `articulo`
  ADD PRIMARY KEY (`codigo`);

--
-- AUTO_INCREMENT de las tablas volcadas
--


--

-- AUTO_INCREMENT de la tabla `articulo`
--
```

```
ALTER TABLE `articulo`
  MODIFY `codigo` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=5;
COMMIT;
```

Instalando node.js

Descargamos de:



Chequeamos la instalación: node -v (en consola)

Creamos primer proyecto:

npm init -y

INSTALAMOS LIBRERIAS ADICIONALES

npm install express --save

npm install mysql

npm install cors

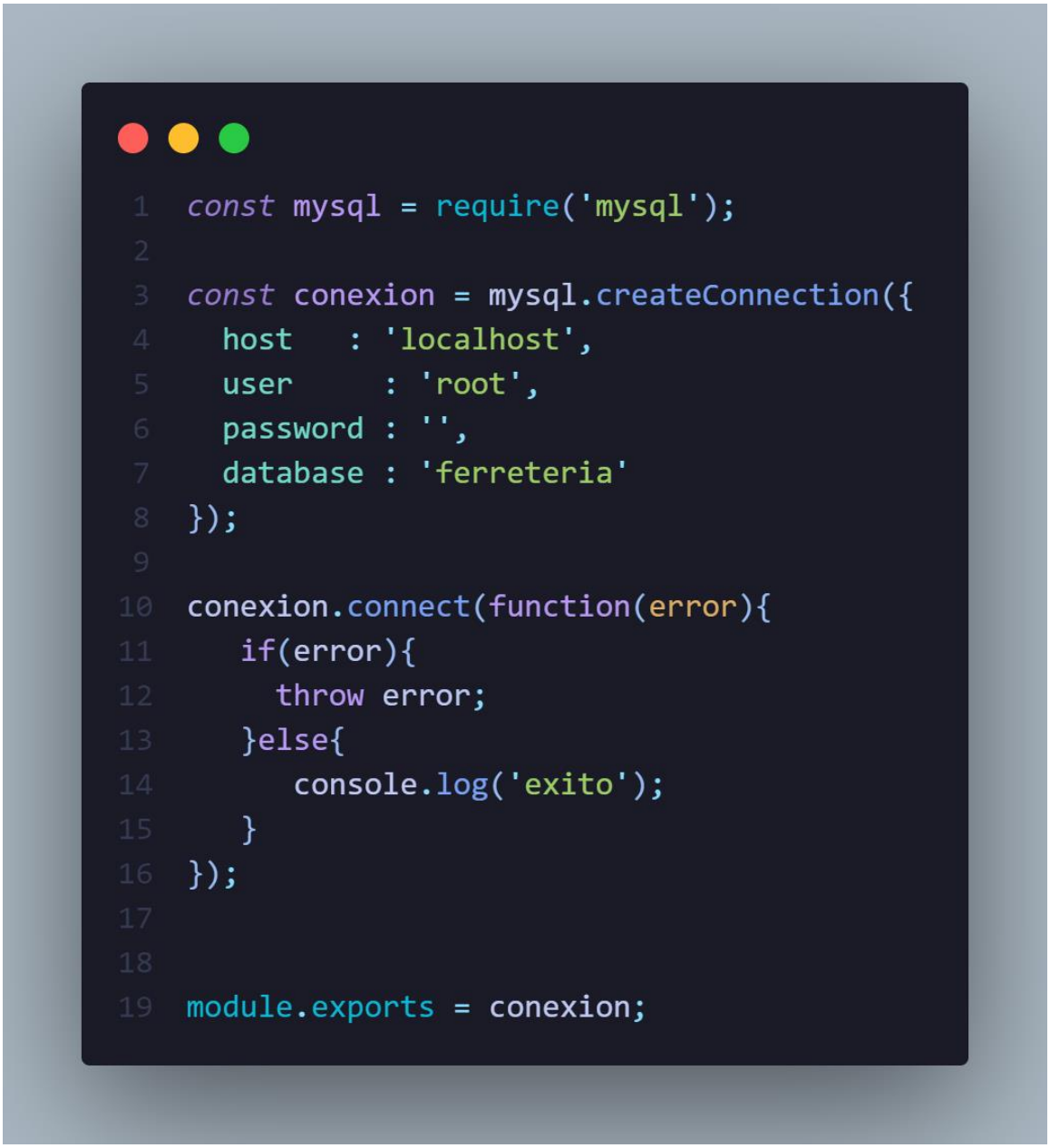
npm install nodemon --save-dev (opcional)

CREACION DE LA API REST

ARCHIVO MAIN

```
const express = require('express');
const main = express();

main.use('/',require('./rutas'));

main.listen('4000', function(){
    console.log('servidor ok en: http://localhost:4000')
});

```

ARCHIVO BD (CONEXIÓN A LA BASE DE DATOS)



```
const mysql = require('mysql');

const conexion = mysql.createConnection({
  host    : 'localhost',
  user      : 'root',
  password : '',
  database : 'ferreteria'
});

conexion.connect(function(error){
   if(error){
     throw error;
   }else{
      console.log('exito');
   }
});


module.exports = conexion;
```

ARCHIVO RUTAS PRIMERA PARTE

```
const express = require('express');
const cors = require('cors');
const ruta = express.Router();

ruta.use(express.json());
ruta.use(cors());

const conexion = require('./bdatos/bd');

ruta.get('/api/articulos/', function (req, res) {

    conexion.query("SELECT * FROM articulo", function (error, resultado) {
        if (error) {
            throw error;
        } else {
            res.send(resultado)
        }
    })
})

module.exports = ruta;
```