



Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en Ciencias y Sistemas

SISTEMAS DE COMUNICACIÓN ASÍNCRONA APLICADOS A LA WEB EN TIEMPO REAL

Erick Carlos Roberto Navarro Delgado

Asesorado por la Inga. Vivian Damaris Campos González

Guatemala, abril de 2016

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA



FACULTAD DE INGENIERÍA

SISTEMAS DE COMUNICACIÓN ASÍNCRONA APLICADOS A LA WEB EN TIEMPO REAL

TRABAJO DE GRADUACIÓN

PRESENTADO A LA JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA

POR

ERICK CARLOS ROBERTO NAVARRO DELGADO

ASESORADO POR LA INGA. VIVIAN DAMARIS CAMPOS GONZÁLEZ

AL CONFERÍRSELE EL TÍTULO DE

INGENIERO EN CIENCIAS Y SISTEMAS

GUATEMALA, ABRIL DE 2016

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERÍA



NÓMINA DE JUNTA DIRECTIVA

DECANO	Ing. Pedro Antonio Aguilar Polanco
VOCAL I	Ing. Angel Roberto Sic García
VOCAL II	Ing. Pablo Christian de León Rodríguez
VOCAL III	Inga. Elvia Miriam Ruballos Samayoa
VOCAL IV	Br. Raúl Eduardo Ticún Córdova
VOCAL V	Br. Henry Fernando Duarte García
SECRETARIA	Inga. Lesbia Magalí Herrera López

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

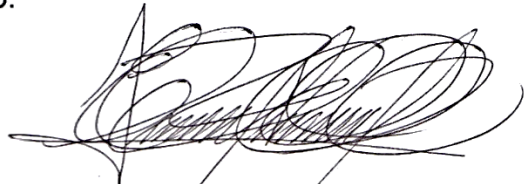
DECANO	Ing. Pedro Antonio Aguilar Polanco
EXAMINADOR	Ing. Oscar Alejandro Paz Campos
EXAMINADOR	Ing. Sergio Arnaldo Méndez Aguilar
EXAMINADOR	Ing. William Estuardo Escobar Argueta
SECRETARIO	Ing. Pablo Christian de León Rodríguez

HONORABLE TRIBUNAL EXAMINADOR

En cumplimiento con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

SISTEMAS DE COMUNICACIÓN ASÍNCRONA APLICADOS A LA WEB EN TIEMPO REAL

Tema que me fuera asignado por la Dirección de la Escuela de Ingeniería en Ciencias y Sistemas, con fecha abril de 2015.

A handwritten signature in black ink, appearing to read 'Erick Carlos Roberto Navarro Delgado', written in a cursive style.

Erick Carlos Roberto Navarro Delgado

Guatemala, 18 de agosto de 2015

Ingeniero
Carlos Azurdia
Escuela de Ciencias y Sistemas
Facultad de Ingeniería
Universidad de San Carlos de Guatemala

Respetable Ingeniero Azurdia:

Por este medio hago de su conocimiento, que como asesora del trabajo de graduación del estudiante de la carrera de Ingeniería en Ciencias y Sistemas, **Erick Carlos Roberto Navarro Delgado**, quien se identifica con el carné **201114595**, hago constar que ha finalizado todos los capítulos del trabajo de investigación titulado: **“Sistemas de comunicación asíncrona aplicados a la web en tiempo real”**, el cual he tenido la oportunidad de revisar y doy mi aprobación al mismo.

Agradeciendo su atención a la presente, me es grato suscribirme.


Inga. Vivian Damaris Campos González de López
Colegiado 5902
Vivian Damaris Campos González
Inga. En Ciencias y Sistemas
Col. 5902



Universidad San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en Ciencias y Sistemas

Guatemala, 3 de Septiembre de 2015

Ingeniero
Marlon Antonio Pérez Türk
Director de la Escuela de Ingeniería
En Ciencias y Sistemas

Respetable Ingeniero Pérez:

Por este medio hago de su conocimiento que he revisado el trabajo de graduación del estudiante **ERICK CARLOS ROBERTO NAVARRO DELGADO** con carné **2011-14595**, titulado: **“SISTEMAS DE COMUNICACIÓN ASÍNCRONA APLICADOS A LA WEB EN TIEMPO REAL”**, y a mi criterio el mismo cumple con los objetivos propuestos para su desarrollo, según el protocolo.

Al agradecer su atención a la presente, aprovecho la oportunidad para suscribirme,

Atentamente,


Ing. Carlos Alfredo Azurdia
Coordinador de Privados
y Revisión de Trabajos de Graduación



E
S
C
U
E
L
A

D
E

I
N
G
E
N
I
E
R
Í
A

E
N

C
I
E
N
C
I
A
S

Y

S
I
S
T
E
M
A
S

UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA



FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA EN
CIENCIAS Y SISTEMAS
TEL: 24188000 Ext. 1534

*El Director de la Escuela de Ingeniería en Ciencias y Sistemas de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer el dictamen del asesor con el visto bueno del revisor y del Licenciado en Letras, del trabajo de graduación **"SISTEMAS DE COMUNICACIÓN ASÍNCRONA APLICADOS A LA WEB EN TIEMPO REAL"**, realizado por el estudiante, **ERICK CARLOS ROBERTO NAVARRO DELGADO** aprueba el presente trabajo y solicita la autorización del mismo.*

"ID Y ENSEÑAD A TODOS"

Ing. Marlon Antonio Pérez Turck
Director

Escuela de Ingeniería en Ciencias y Sistemas



Guatemala, 31 de marzo de 2016

Universidad de San Carlos
de Guatemala



Facultad de Ingeniería
Decanato

Ref.DTG.D.144.2016

El Decano de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer la aprobación por parte del Director de la Escuela de Ingeniería en Ciencias y Sistemas, al trabajo de graduación titulado: **SISTEMAS DE COMUNICACIÓN ASÍNCRONA APLICADOS A LA WEB EN TIEMPO REAL**, presentado por el estudiante universitario: **Erick Carlos Roberto Navarro Delgado**, y después de haber culminado las revisiones previas bajo la responsabilidad de las instancias correspondientes, se autoriza la impresión del mismo.

IMPRÍMASE.


Ing. Pedro Antonio Aguilar Polanco
Decano

Guatemala, abril de 2016



/cc

ACTO QUE DEDICO A:

- Dios** Por iluminar en todo momento mi camino, ser fuente de fortaleza en mi vida, permitirme avanzar en mi formación académica y llenarme de tantas bendiciones.
- Virgen María** Por ser la estrella que me mostró el camino en mis noches oscuras y mi mejor consuelo en los momentos difíciles.
- Mi madre** Sandra Delgado, por su amor, esfuerzos y sacrificios. No tengo palabras para agradecer todo lo que has hecho por mí.
- Mis hermanos** Joaquín y Estuardo Navarro, por motivarme a seguir adelante y acompañarme en las buenas y en las malas.
- Mis abuelos** Carmen García y Carlos Delgado, por su compañía, comprensión, cariño y apoyo. Por estar siempre conmigo.
- Mi tío** Guillermino Mejía, por enseñarme tantas cosas importantes y apoyarme en todo momento.

Mi padrino

Antonio Delgado, por su apoyo a lo largo de mi carrera, sus valiosos consejos y por ayudarme a salir adelante.

Mi familia

Por compartir conmigo los buenos y los malos momentos, por alegrarse de mis triunfos y apoyarme incondicionalmente.

Mis amigos

Por ser mis compañeros de batalla, con quienes he caminado durante estos años de carrera universitaria y con quienes espero seguir caminando. En especial a Ana Lucía López, Carlos Yoque, Diego Solis y Hugo González.

AGRADECIMIENTOS A:

Guatemala	La tierra que me vio crecer, espero que mi profesión y mis conocimientos sean útiles para tu progreso.
Universidad de San Carlos de Guatemala	Por ser mi casa de estudios, la universidad del estado que me abrió sus puertas para ser uno más de sus alumnos.
Facultad de Ingeniería	Por haberme brindado tantos conocimientos y experiencias de vida.
Mi asesora	Inga. Damaris Campos de López, por compartir conmigo sus conocimientos y por su apoyo en la elaboración de este trabajo.

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES	V
LISTA DE SÍMBOLOS	VII
GLOSARIO	IX
RESUMEN	XIII
OBJETIVOS	XV
INTRODUCCIÓN	XVII
1. CONCEPTOS TEÓRICOS GENERALES	1
1.1. La web en tiempo real	1
1.1.1. Sistemas de tiempo real	1
1.1.1.1. Clasificación.....	2
1.1.1.2. Propiedades importantes	3
1.1.1.3. Características.....	4
1.1.2. Aplicaciones web	5
1.1.3. Aplicaciones web <i>versus</i> aplicaciones de escritorio	5
1.1.4. ¿Qué es la web en tiempo real?	7
1.2. Sistemas de comunicación	8
1.2.1. Sistemas de comunicación síncronos.....	8
1.2.2. Sistemas de comunicación asíncronos.....	10
2. AJAX, JAVASCRIPT ASÍNCRONO Y XML	13
2.1. Modelo clásico de aplicaciones web (síncrono).....	14
2.2. Modelo Ajax de aplicaciones web (asíncrono).....	14
2.3. Tecnologías que alberga el concepto de Ajax.....	18

2.3.1.	JavaScript.....	18
2.3.2.	JSON.....	19
2.3.3.	XMLHttpRequest.....	22
2.3.4.	XML.....	23
2.3.5.	XSLT.....	24
2.3.6.	DOM.....	25
2.3.7.	XHTML.....	27
2.3.8.	CSS.....	28
2.4.	Modelo de eventos.....	28
2.4.1.	Flujo de eventos.....	29
2.4.2.	Manejadores de eventos.....	30
2.5.	Marcos de trabajo y librerías complementarias.....	30
2.5.1.	Prototype.....	31
2.5.2.	jQuery.....	31
2.5.3.	Mootools.....	32
2.6.	¿Quién está utilizando Ajax?.....	32
2.7.	El papel de Ajax en la web en tiempo real.....	33
3.	NODE.JS, UN INTÉRPRETE ASÍNCRONO PARA JAVASCRIPT DEL LADO DEL SERVIDOR.....	35
3.1.	¿Qué es Node.js?.....	35
3.2.	¿Cómo funciona Node.js?.....	37
3.3.	El motor V8 de JavaScript.....	39
3.4.	NPM.....	39
3.5.	Marco de trabajo monohilo, asíncrono y dirigido por eventos ..	40
3.6.	El modelo no bloqueante de entrada y salida.....	42
3.7.	Módulo Socket.IO y Websockets.....	42
3.8.	El papel de Node.js en la web en tiempo real.....	43
3.9.	Pruebas de rendimiento, Node.js <i>versus</i> Apache.....	47

4.	CASO DE ESTUDIO	51
4.1.	Descripción del caso de estudio	51
4.2.	Requerimientos	52
4.2.1.	Gestión de los clientes.....	52
4.2.2.	Selección de subasta.....	53
4.2.3.	Gestión de subastas.....	54
4.2.4.	Participación en la subasta.....	54
4.3.	Solución propuesta.....	55
4.3.1.	Inicio de sesión.....	55
4.3.2.	Selección de subasta.....	57
4.3.3.	Gestión de subastas.....	60
4.3.4.	Monitor de subasta	64
	4.3.4.1. Solución a los posibles problemas de concurrencia	66
	CONCLUSIONES	69
	RECOMENDACIONES.....	71
	BIBLIOGRAFÍA.....	73
	APÉNDICES	75

ÍNDICE DE ILUSTRACIONES

FIGURAS

1.	Ilustración del diseño web responsivo.....	7
2.	Comunicación síncrona del modelo tradicional de aplicaciones web....	9
3.	Comunicación asíncrona de una emisión televisiva.....	11
4.	El modelo tradicional para aplicaciones web comparado con el modelo Ajax	16
5.	Patrón de interacción síncrona de una aplicación web tradicional comparado con el patrón asíncrono de una aplicación Ajax	17
6.	Fracción de código en JSON con información de un inventario.....	21
7.	Equivalente gráfico del código mostrado en la figura 6.....	22
8.	Código XML equivalente al código JSON mostrado en la figura 6.....	24
9.	Código XHTML ejemplo	26
10.	Ejemplo de estructura de datos generada por DOM	27
11.	Ejemplo de Google Suggest ejecutándose	33
12.	Gráfica del bucle de eventos de Node.js.....	38
13.	Posición en el mercado de diferentes servidores web	45
14.	Popularidad de Node.js <i>versus</i> Apache	46
15.	Código PHP utilizado en pruebas de rendimiento.....	48
16.	Código Node.js utilizado en pruebas de rendimiento	48
17.	Diagrama de estados de una subasta.....	54
18.	<i>Mockup</i> de la página de inicio de sesión visualizada en un teléfono inteligente.....	56
19.	<i>Mockup</i> de la página de selección de subastas visualizada en un teléfono inteligente	58

20.	<i>Mockup</i> de la página de selección de subastas al seleccionar una subasta en estado activo, visualizada en un teléfono inteligente.....	59
21.	<i>Mockup</i> de la página de selección de subastas al seleccionar una subasta en estado inactivo, visualizada en un teléfono inteligente.....	60
22.	<i>Mockup</i> de la página de gestión de subastas, visualizada en una tableta digital.....	62
23.	<i>Mockup</i> de la página de gestión de subastas con algunas subastas desplegadas, visualizada en una tableta digital.....	63
24.	<i>Mockup</i> de la página del monitor de subasta, visualizada en una tableta digital.....	65
25.	<i>Mockup</i> del mensaje de advertencia mostrado a los usuarios cuando intentan participar más de una vez en una subasta.....	67

TABLAS

I.	Cuadro comparativo de sistemas comunicación síncrona <i>versus</i> sistemas comunicación asíncrona.....	12
II.	Especificaciones de software y hardware de la computadora utilizada en las pruebas de rendimiento.....	47
III.	Resultados de las pruebas de rendimiento de Apache.....	49
IV.	Resultados de las pruebas de rendimiento de Node.js.....	49
V.	Especificaciones de software y hardware del servidor utilizado por la empresa de subastas.....	52

LISTA DE SÍMBOLOS

Símbolo	Significado
B	Byte
GB	Gigabyte
GHz	Gigahercio
KB	Kilobyte
km	Kilometro
ms	Milisegundo
#	Número
%	Porcentaje
s	Segundo

GLOSARIO

ApacheBench	Es una herramienta de evaluación comparativa perteneciente a las utilidades del servidor HTTP Apache.
Bucle	Es una secuencia de acciones que se ejecutan repetidamente mientras se cumpla cierta condición.
C++	Lenguaje de programación orientado a objetos proveniente de C.
Captcha	Son las siglas de <i>Completely automated public turing test to tell computers and humans apart</i> , en español, prueba de Turing completamente automática y pública para diferenciar computadoras de humanos. Es llamada también prueba de Turing inversa, por ser controlada por una máquina en lugar de un humano.
Cliente	Es una aplicación que consume un servicio brindado por otra herramienta de software conocida como servidor.
DBMS	Es un sistema gestor de bases de datos que sirve como interfaz entre el usuario, otras aplicaciones y la base de datos.

Google	Empresa multinacional estadounidense que desarrolla productos relacionados con las tecnologías de la información y la comunicación.
<i>Handshake</i>	Negociación que se lleva a cabo para establecer la comunicación entre cliente y servidor.
HTML	<i>Hypertext markup language</i> . Es un lenguaje de marcas utilizado para la elaboración de páginas web.
HTTP	<i>Hypertext transfer protocol</i> . Es el protocolo utilizado por la web para realizar todas sus transacciones.
Internet	Es una red de acceso público que interconecta muchas redes de computadoras a nivel mundial y transmite información utilizando IP, el protocolo de internet.
LTS	<i>Long term support</i> . Es el soporte técnico extendido de algunas versiones de Ubuntu que son liberadas cada dos años.
MD5	Algoritmo de resumen del mensaje 5, es un algoritmo de reducción criptográfica ampliamente utilizado.
<i>Mockup</i>	Es un modelo a escala de un diseño utilizado para la demostración, evaluación, promoción y otros fines.

MySQL	Es un sistema de gestión de bases de datos relacional de código abierto.
PHP	Lenguaje de programación diseñado para el desarrollo de aplicaciones web.
Servidor	Software que posee la capacidad de atender las peticiones de uno o más clientes y devolver una respuesta ante dicha petición.
<i>Thread pool</i>	Concepto propio de Node.js que se refiere a un grupo de subprocesos que se ejecutan asíncronamente.
<i>Timeout</i>	Evento que sucede cuando el tiempo de espera para cierta conexión vence y se termina con el proceso que esperaba dicha conexión.
URL	<i>Uniform resource locator</i> . Es un localizador de recursos uniforme, el recurso al que dicho localizador apunta puede variar a lo largo del tiempo.
W3C	<i>World wide web consortium</i> . Es una comunidad internacional que desarrolla estándares que procuren el crecimiento de la web.

RESUMEN

Las aplicaciones web han evolucionado rápidamente en los últimos años, principalmente desde el surgimiento de las redes sociales. Esta evolución ha requerido nuevas soluciones para manejar grandes cantidades de información en lapsos reducidos. Como respuesta a estas necesidades de la web, surgió el concepto de la web en tiempo real, el cual reúne un conjunto de tecnologías que buscan mejorar el rendimiento de las aplicaciones para obtener los resultados esperados en el menor tiempo. Uno de los elementos más importantes de la web en tiempo real son los sistemas de comunicación asíncrona, con los que se obtienen considerables mejoras en el rendimiento comparado con los sistemas de comunicación síncronos usados en el modelo clásico de las aplicaciones web.

Ajax es un concepto que reúne un conjunto de tecnologías con las que se pueden crear aplicaciones web que funcionan de manera muy agradable para el usuario, porque permiten una interacción fluida con el servidor, debido al sistema de comunicación asíncrona que implementa entre el cliente y el servidor, que tiene como intermediario el motor de Ajax.

Node.js es una tecnología que puede utilizarse para implementar sistemas de comunicación asíncrona en aplicaciones web a través de su módulo Socket.IO, que permite utilizar *sockets* para el intercambio de información. Además, su funcionamiento asíncrono y con un solo hilo de ejecución permite obtener muy buenos resultados de rendimiento. Lo anterior hace de Node.js una herramienta útil en el desarrollo de aplicaciones web en tiempo real.

OBJETIVOS

General

Realizar un análisis de los sistemas de comunicación asíncrona, basados en *sockets* y tecnologías Ajax, utilizados en los sitios web, destacando las ventajas que estos tienen sobre los sistemas de comunicación síncrona utilizados tradicionalmente, demostrando que son la mejor opción en la creación de sitios web que presentan la información al usuario en tiempo real.

Específicos

1. Demostrar que un sistema de comunicación asíncrona es mejor que un sistema de comunicación síncrona para el desarrollo de un sitio web en tiempo real.
2. Proporcionar una recopilación de las características más relevantes de Ajax y Node.js, tecnologías que son claro ejemplo de los beneficios que brindan los sistemas de comunicación asíncrona.
3. Realizar un análisis de cómo las tecnologías web pueden ajustarse a las necesidades de un sistema de tiempo real.

INTRODUCCIÓN

Las aplicaciones web son de suma importancia en el mundo de las tecnologías de la información, porque presentan muchas ventajas sobre las tradicionales aplicaciones de escritorio. En los últimos años, las redes sociales se han popularizado considerablemente y, con ellas, las aplicaciones web han evolucionado, ofreciendo nuevas formas de manejar la información de sistemas que exigen cada vez más, porque la cantidad de información que manejan crece exponencialmente, así como el número de usuarios que desean acceder a esta información. Dentro de esta evolución de las tecnologías web, surge el innovador concepto de la web en tiempo real.

La web en tiempo real permite a los usuarios acceder a la información inmediatamente después de que esta se genera, lo que significa que el lapso entre el momento en el que la información se genera y en el que el usuario recibe esa información debe ser sumamente pequeño. Esto representa un gran reto para las tecnologías web tradicionales que utilizan un sistema de comunicación síncrona entre el cliente y el servidor, en el que cada vez que el usuario desea obtener información del servidor, tiene que realizar una petición y esperar una respuesta por parte del servidor. Para esto, es necesario recargar la página web en el navegador y hacerlo representa un consumo de tiempo, en respuesta a este problema se implementaron sistemas asíncronos en las tecnologías web. En el presente trabajo se presentan Node.js y Ajax como dos sistemas asíncronos que pueden utilizarse para lograr una web en tiempo real.

Node.js es una tecnología basada en el motor V8 de JavaScript de Google, este motor está diseñado para correr en un navegador y se

caracteriza por la alta velocidad a la que puede interpretar código de JavaScript. Node.js utiliza JavaScript del lado del servidor, haciendo esto, logra resultados de rendimiento impresionantes.

Ajax (JavaScript asíncrono y XML) es un conjunto de tecnologías que se utilizan para obtener información del servidor o enviar información al servidor en segundo plano sin interferir con la visualización o el comportamiento de la página, es JavaScript el que se encarga de gestionar los datos asíncronos obtenidos.

1. CONCEPTOS TEÓRICOS GENERALES

Este capítulo abarca el contenido teórico relacionado con la web en tiempo real y los sistemas de comunicación, es necesario conocer todo este fundamento teórico para tener un panorama claro de la integración de los sistemas de comunicación asíncrona con la web en tiempo real y los beneficios que dicha integración puede proporcionar.

1.1. La web en tiempo real

La web en tiempo real ofrece al usuario la información inmediatamente después de que se genera, en un entorno web. Se ha popularizado ampliamente porque la satisfacción del usuario incrementa cuando obtiene resultados en menor tiempo, además, hay sistemas que exigen obtener la información en tiempo real, porque es así como funcionan. Existe una amplia teoría detrás de los sistemas de tiempo real y esta teoría debe utilizarse como base en la construcción de la web en tiempo real.

1.1.1. Sistemas de tiempo real

Los sistemas de tiempo real son aquellos en los que la obtención de los resultados está sujeta a restricciones temporales que son impuestas por el entorno en el que se ejecutan¹.

¹ *Conceptos básicos en los sistemas de tiempo real.*
http://www.uv.es/gomis/Apuntes_SITR/Conceptos.pdf. Consulta: marzo de 2015.

Las restricciones temporales de los sistemas de tiempo real generalmente consideran lapsos muy cortos, esto requiere que el procesamiento y la transmisión de información se realicen a altas velocidades, las cuales satisfagan las restricciones de tiempo.

Los sistemas de tiempo real no solo tienen que brindar resultados correctos, también tienen que respetar el tiempo límite contemplado en la restricción temporal.

Una emisión radial en vivo es un buen ejemplo de sistema de tiempo real. En este sistema, la voz del locutor debe llegar a las personas que están escuchando la emisión en un lapso muy corto, lo suficientemente corto como para dar la impresión a los oyentes de que reciben la información justo en el instante en que se genera, es decir, ellos tienen la impresión de que escuchan al locutor justo cuando habla, escuchan al locutor en tiempo real. Por supuesto que existe un pequeño retardo entre el momento en el que el locutor habla y el momento en el que los oyentes escuchan lo que dijo, pero dicho retardo es mínimo y se ajusta a las restricciones temporales. Las transmisiones televisivas en vivo también son un ejemplo de sistema de tiempo real.

1.1.1.1. Clasificación

Existen muchas clasificaciones de los sistemas de tiempo real, pero la más importante de ellas es la que se basa en el nivel de exigencia de las restricciones temporales, según esta clasificación existen cuatro tipos de sistemas:

- Críticos: son aquellos sistemas en los que las restricciones de tiempo tienen que cumplirse siempre, porque de no ser así las consecuencias

pueden ser fatales. Un ejemplo puede ser un sistema que controla el tráfico aéreo en tiempo real, si no se obtiene la información sobre la ruta de los aviones a tiempo, puede producirse un accidente aéreo.

- **Esenciales:** son los sistemas en los que deben cumplirse las restricciones de tiempo, si se incumplieran las consecuencias no serían fatales pero el sistema dejaría de funcionar correctamente y el usuario no obtendría los resultados deseados. Por ejemplo, si un sistema de videoconferencia, incumple con la restricción temporal, puede que los participantes de la videoconferencia se atrasen en sus labores.
- **Incrementales:** en estos sistemas, la calidad de la respuesta aumenta conforme se le da más tiempo al sistema para procesarla. Por ejemplo, si se tiene un sistema que juega ajedrez con el usuario, el movimiento que decida hacer el sistema depende del tiempo que se le haya dado para procesar la información, a mayor tiempo de procesamiento, mayor calidad de respuesta.
- **No esenciales:** no están sujetos a restricciones temporales, simplemente tiene que ejecutarse en un corto tiempo, lo más rápido posible.

1.1.1.2. Propiedades importantes

Un sistema de tiempo real cuenta con ciertas propiedades que definen su comportamiento con relación a las restricciones temporales, estas propiedades son:

- **Garantía de plazos:** se dice que un sistema de tiempo real funciona correctamente cuando todas las tareas se realizan dentro del plazo

determinado por las restricciones temporales, todas las tareas realizan sus actividades respetando los límites de tiempo.

- Tiempo de respuesta máximo: en los sistemas de tiempo real se procura establecer el tiempo límite de las tareas basándose en el peor de los casos, es decir, lo más que puede tardar una tarea es lo que tarda en el peor de los casos, ese es el tiempo límite. Se trata de que las tareas se realicen en un tiempo que sea lo más cercano posible al tiempo que se tarda el sistema en realizar la tarea en el mejor de los casos, es decir, el tiempo más corto.
- Estabilidad: si el sistema llegara a un estado en el que no puede realizar todas las tareas que se le solicitan dentro del plazo temporal establecido, se debe garantizar que por lo menos un subgrupo de las tareas se realice a tiempo, estas tareas deben ser las que se consideran con mayor prioridad, las tareas críticas.

1.1.1.3. Características

Las características más importantes de un sistema de tiempo real son las siguientes:

- Determinismo: esta característica consiste en conocer exactamente cómo se comporta el entorno del sistema, es decir, el conjunto de elementos que producen información que sirve como entrada al sistema. De tal manera que se tiene la certeza de que no existirán situaciones en las que el sistema no puede responder a las peticiones porque las entradas no son las adecuadas.

- Comportamiento predecible: saber cómo se comporta el sistema y cómo se comportará, para tener la certeza de que no aparecerán situaciones que alteren el comportamiento temporal del sistema. Por ejemplo, si el sistema tiene un comportamiento predecible se sabrá que nunca tendrá entradas que provoquen un retraso en el tiempo de procesamiento, es decir, entradas que para ser procesadas necesiten un tiempo mayor que el tiempo máximo establecido en las restricciones temporales.

1.1.2. Aplicaciones web

Son herramientas de software que se utilizan a través de un navegador que funciona como cliente y realiza peticiones a un servidor web, es necesario que el dispositivo en el que se ejecuta el navegador esté conectado al servidor web a través de una red.

1.1.3. Aplicaciones web *versus* aplicaciones de escritorio

Las aplicaciones web cuentan con una serie de ventajas sobre las aplicaciones de escritorio y es por ello que en los últimos años se ha buscado desarrollar en entorno web, los programas que antiguamente se ejecutaban como aplicaciones de escritorio. Un ejemplo bastante popular de esta tendencia es el caso de las herramientas de Google Docs, que cuentan con un procesador de texto, una herramienta para el uso de hojas de cálculos y otra para la creación de presentaciones con diapositivas. Estas herramientas aún están muy por debajo de sus equivalentes en aplicaciones de escritorio, pero se desarrollan constantemente y están encaminadas a imitar, de la mejor forma posible, las funciones de las aplicaciones de escritorio.

A continuación se describen algunas de las principales ventajas que las aplicaciones web poseen sobre las aplicaciones de escritorio:

- Para acceder a una aplicación web solamente se instala un navegador de páginas web en el cliente. Básicamente es lo único que se requiere del lado del cliente, no hace falta instalar software especializado, aunque puede darse el caso de que se necesite algún otro software complementario, dependiendo de la tarea que la aplicación web deba realizar, pero la cantidad de software a instalar es mínima.
- La actualización de los equipos con una nueva versión representa un bajo costo comparado con lo que costaría actualizar aplicaciones de escritorio, porque, como mucho, deberían instalarse las actualizaciones del navegador y los complementos que se necesiten para el buen funcionamiento de la aplicación web, estas actualizaciones no son tan significativas comparadas con las que requeriría una aplicación de escritorio. El navegador en los clientes solamente muestra en pantalla lo que el servidor le envía, esto significa que la actualización principal solo debe hacerse en el servidor, una vez actualizado el servidor, todos los clientes que accedan desde sus navegadores verán la aplicación actualizada. Con esto también se asegura de que todos los usuarios utilizarán la última versión, siempre que el cliente esté configurado adecuadamente.
- Una ventaja muy importante es la movilidad, si el software está ubicado en un servidor web en internet, cualquier usuario con acceso a internet y un dispositivo con navegador podrá acceder a la aplicación, sin importar el lugar en el que esté podrá hacer uso de la aplicación web. Además, el diseño web responsivo permite desarrollar una sola aplicación que puede

usarse en múltiples dispositivos móviles de manera cómoda, porque las páginas web se adaptan al tamaño de la pantalla en la que se utilizan. En la figura 1 se observa una ilustración del diseño web responsivo.

Figura 1. **Ilustración del diseño web responsivo**



Fuente: *Diseño web responsive adaptable a cualquier dispositivo smartphone, tablet, PC o TV.*
<http://adnstudio.com/disenio-web-responsive-adaptable-a-cualquier-dispositivo-smartphone-tablet-pc-o-tv/>. Consulta: marzo de 2015

1.1.4. ¿Qué es la web en tiempo real?

Las aplicaciones web han crecido considerablemente y las demandas de dichas aplicaciones han crecido con ellas. La cantidad de información que las aplicaciones web manejan es cada vez mayor y los usuarios desean obtener esta información en tiempos cortos. En respuesta a esto surge la web en tiempo real, un concepto innovador en el que el usuario obtiene la información

inmediatamente después de que esta se genera. El modelo clásico de las aplicaciones web presenta algunas deficiencias en el tema del tiempo, principalmente porque el tipo de comunicación que se maneja entre el cliente y el servidor es síncrona.

Para la web en tiempo real, el tema del tiempo es muy importante, porque el procesamiento y la transmisión de información debe ser lo suficientemente rápida como para darle al usuario la impresión de que obtiene la información inmediatamente después de que se genera. Una considerable mejora en el tiempo de respuesta de las aplicaciones web es la aplicación de sistemas de comunicación asíncrona.

1.2. Sistemas de comunicación

Un sistema de comunicación es aquel que tiene como principal función transmitir información desde un sistema emisor hasta un sistema receptor, mediante un canal de comunicación.

En informática y telecomunicaciones, los sistemas de comunicación se dividen en dos grandes grupos según la forma en que el emisor interactúa con el receptor, estos grupos son los sistemas de comunicación síncronos y los sistemas de comunicación asíncronos.

1.2.1. Sistemas de comunicación síncronos

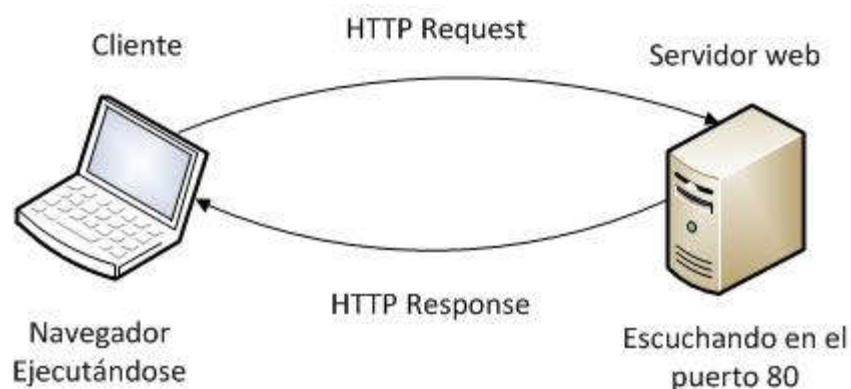
En los sistemas de comunicación síncronos, el emisor espera siempre una respuesta del receptor luego de haber enviado el mensaje. El hecho de que espere una respuesta del receptor implica, en la mayor parte de los casos, que no puede establecer comunicación con ningún otro sistema a través de ese

canal hasta que deje de esperar, es decir, hasta que obtenga una respuesta por parte del receptor.

Un ejemplo sencillo de comunicación síncrona es una llamada telefónica, el emisor realiza la llamada y debe esperar a que el receptor responda, es decir, acepte la llamada. Luego de que el emisor se coordina con el receptor, se establece la conexión e inicia el intercambio de información a través del canal de comunicación.

En el modelo tradicional de las aplicaciones web, se hace uso de un sistema de comunicación síncrona, ya que el cliente realiza una petición al servidor web, espera hasta que el servidor procese dicha petición y la responda con una nueva página HTML que se imprimirá en el navegador del usuario, como se observa en la figura 2.

Figura 2. **Comunicación síncrona del modelo tradicional de aplicaciones web**



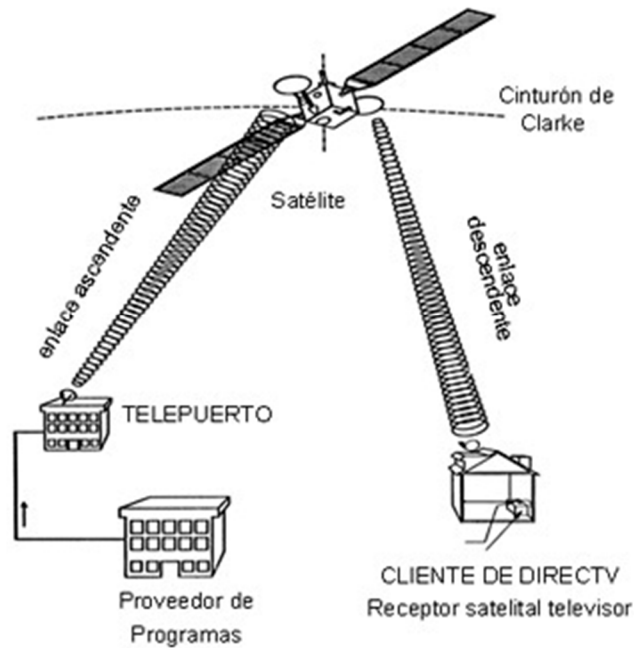
Fuente: elaboración propia, empleando Microsoft Visio 2010.

1.2.2. Sistemas de comunicación asíncronos

En los sistemas de comunicación asíncrona, el emisor envía el mensaje y no espera la respuesta del receptor, esto significa que el emisor puede volver a utilizar libremente el canal de comunicación inmediatamente después de haber enviado el mensaje.

Un ejemplo de comunicación asíncrona que resulta familiar para la mayoría de personas es la televisión, esta utiliza un sistema de comunicación asíncrona en el que la cadena televisiva, que funciona como emisor, envía constantemente información a través de señales al televisor, que funciona como receptor. Luego de que el televisor capta las señales, muestra la información que recibe en pantalla y lo sigue haciendo mientras este encendido. Se dice que es un sistema de comunicación asíncrona porque la cadena televisiva, que es el emisor, no espera respuesta alguna del receptor, que es el televisor, el flujo de la comunicación se muestra en la figura 3.

Figura 3. **Comunicación asíncrona de una emisión televisiva**



Fuente: *Preguntas frecuentes sobre el sistema de televisión satelital.*

<http://www.directv.com.pe/servicio-al-cliente/como-funciona>. Consulta: marzo de 2015.

Retomando el ejemplo de comunicación síncrona de la llamada telefónica, si la persona que llama no obtiene respuesta del receptor y decide dejarle un mensaje cuando se active el correo de voz, entonces el sistema de comunicación deja de ser síncrono y pasa a ser asíncrono, porque el emisor luego de dejar el mensaje en el correo de voz, no espera respuesta alguna del receptor. El mensaje de voz simplemente se almacena en algún lugar y posteriormente es escuchado por el receptor.

En la tabla I se muestra un cuadro comparativo que resume las ventajas y desventajas de los sistemas de comunicación que fueron descritos anteriormente.

Tabla I. **Cuadro comparativo de sistemas comunicación síncrona versus sistemas comunicación asíncrona**

Sistemas de comunicación síncrona	
Ventajas	El emisor tiene la certeza de que el receptor ha recibido el mensaje satisfactoriamente, cuando recibe la respuesta.
	Se conoce con certeza el orden en el que el receptor recibe los mensajes.
Desventajas	El emisor no puede enviar un nuevo mensaje hasta que el receptor haya respondido el último mensaje.
	El tiempo que el emisor espera una respuesta disminuye la velocidad del proceso comunicativo.
Sistemas de comunicación asíncrona	
Ventajas	El emisor está listo para enviar un nuevo mensaje justo después de haber enviado el último mensaje.
	El enviar mensajes consecutivos sin esperar respuesta permite transmitir mucha información en poco tiempo.
Desventajas	Es necesario hacer validaciones adicionales para verificar que el receptor recibió el mensaje.
	Puede que dos o más mensajes no lleguen al receptor en el orden en que fueron enviados, por el tamaño del mensaje, por ejemplo.

Fuente: elaboración propia.

2. AJAX, JAVASCRIPT ASÍNCRONO Y XML

Ajax es un término que fue utilizado por primera vez por Jesse James Garret, en un artículo publicado en febrero de 2005. Es el acrónimo para JavaScript asíncrono y XML, Garret define Ajax, no como una tecnología, sino como muchas tecnologías, cada una floreciendo por mérito propio, uniéndose en poderosas nuevas formas².

La idea principal de Ajax es cargar inicialmente una página completa, luego hay un conjunto de *scripts* y procesos que van en busca de información al servidor. Cuando estos *scripts* obtienen la información, se modifica la página, actualizándose solo algunos segmentos de la misma, o si se desea, puede actualizarse la página completa. Esto significa que es posible alterar el contenido de la página sin tener que recargarla y se pueden realizar peticiones al servidor sin necesidad de recargar toda la página.

La idea es bastante simple, pero esta simple solución hace que las aplicaciones web se acerquen más a las aplicaciones de escritorio, porque con Ajax se logra esa interacción y ese tiempo de respuesta que antes parecía inalcanzable para las aplicaciones web; todo esto se obtiene gracias al modelo asíncrono que se plantea. Con Ajax, el diseño de las páginas web mejora considerablemente, así como la interacción entre el usuario y la aplicación web, esto abre las puertas a muchas posibilidades para nuevas aplicaciones web que pueden desarrollarse a partir de este nuevo concepto.

² GARRET, Jesse James. *Ajax: A new approach to web applications*. <http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications/>. Consulta: marzo de 2015.

2.1. Modelo clásico de aplicaciones web (síncrono)

En el modelo clásico de aplicaciones web, cada vez que el usuario ejecuta una acción, por ejemplo el envío de un formulario lleno, se dispara una petición HTTP. Cuando esta petición llega al servidor, este la procesa, ejecuta las acciones relacionadas con la petición y envía como respuesta al cliente una página HTML que el navegador mostrará al usuario.

Este modelo implementa un sistema de comunicación síncrona porque el emisor, en este caso el cliente, luego de enviar el mensaje espera una respuesta del receptor, en este caso el servidor. Del lado izquierdo de la figura 4 se observa una ilustración del modelo tradicional de aplicaciones web. En la parte de arriba de la figura 5 se ilustra la interacción entre el cliente y el servidor en el sistema de comunicación síncrona.

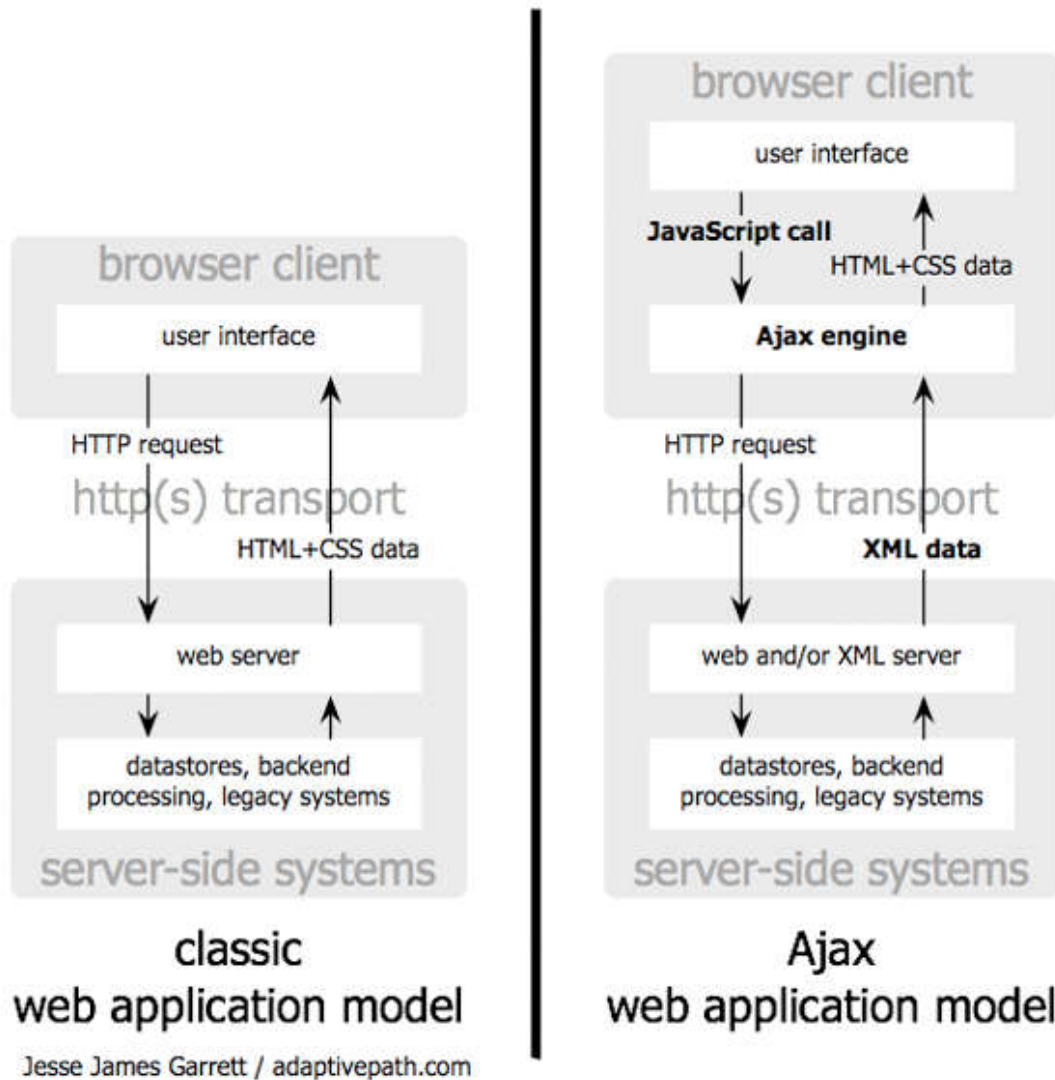
2.2. Modelo Ajax de aplicaciones web (asíncrono)

El modelo clásico funciona porque se realizan las tareas que se necesitan realizar, el problema es que cada vez que el cliente desea interactuar con el servidor, tiene que esperar a que sus datos lleguen al servidor, que el servidor los procese y que la respuesta del servidor vuelva. A los usuarios no les agrada esperar, especialmente si se trata de un sistema de tiempo real, en el que el tiempo es un factor crucial.

El modelo Ajax resuelve estos problemas de espera con un sistema asíncrono que utiliza un motor Ajax entre el cliente y el servidor. Este motor permite que la comunicación se haga asíncronamente porque es independiente del servidor. Con esto se logra que el usuario ya no tenga que ver la pantalla en blanco del navegador mientras espera la respuesta del servidor y, por lo tanto,

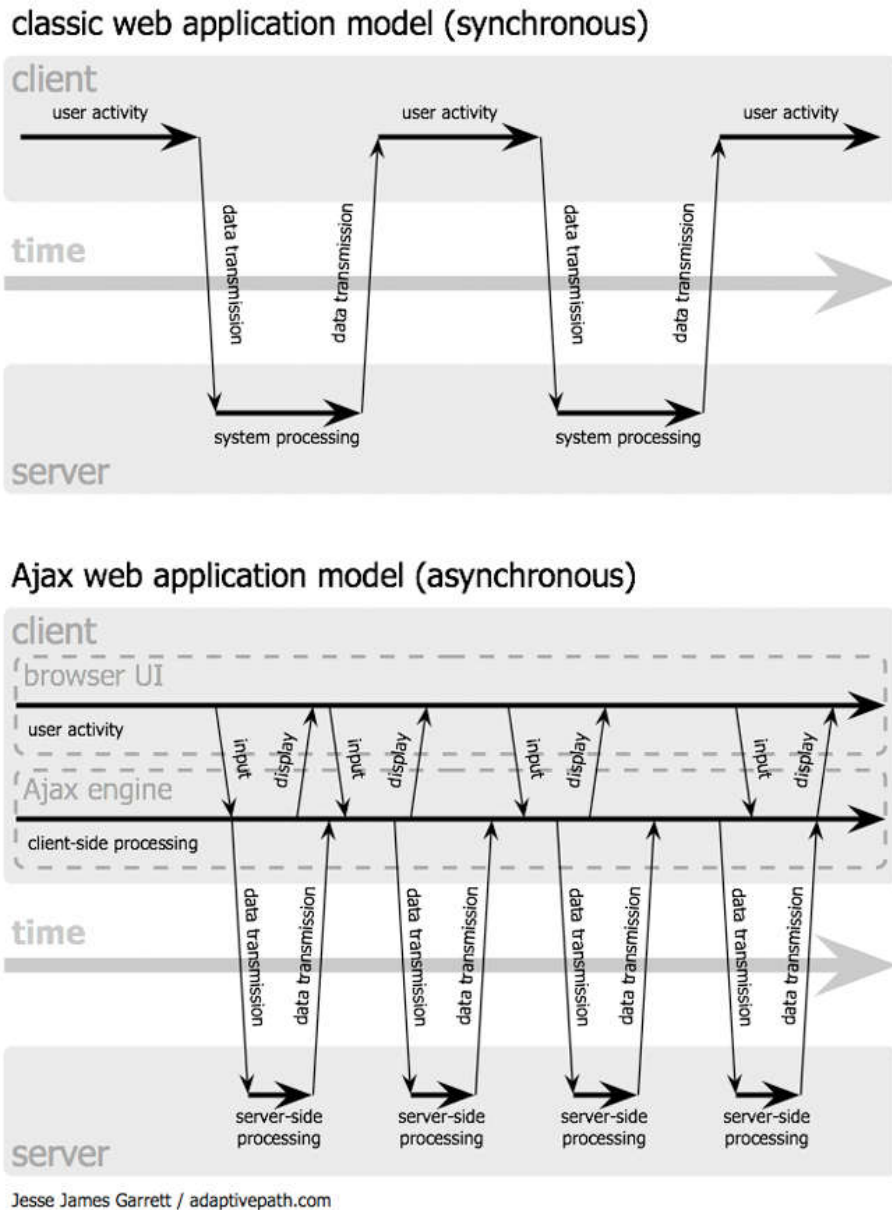
la satisfacción del usuario se incrementa y el tiempo de respuesta mejora considerablemente. Del lado derecho de la figura 4 se observa una ilustración del modelo Ajax. En la parte de abajo de la figura 5 se ilustra la interacción entre el cliente y el servidor en el sistema de comunicación asíncrona, teniendo como intermediario al motor de Ajax.

Figura 4. El modelo tradicional para aplicaciones web comparado con el modelo Ajax



Fuente: *Ajax: A new approach to web applications*. <http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications/>. Consulta: marzo de 2015.

Figura 5. Patrón de interacción síncrona de una aplicación web tradicional comparado con el patrón asíncrono de una aplicación Ajax



Fuente: *Ajax: A new approach to web applications*. <http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications/>. Consulta: marzo de 2015.

2.3. Tecnologías que alberga el concepto de Ajax

Como se mencionó anteriormente, Ajax está formado por muchas tecnologías. En su artículo sobre Ajax, Jesse Garret dice que el concepto de Ajax incorpora lo siguiente:

- Una presentación basada en estándares usando XHTML y CSS
- Exhibición e interacción dinámicas usando el DOM
- Intercambio y manipulación de datos usando XML y XSLT
- Recuperación de datos asíncrona usando XMLHttpRequest
- JavaScript que reúne a todos los anteriores

Todas estas tecnologías se integran para lograr la interacción deseada entre el usuario y la aplicación web, es importante entender estas tecnologías por separado, para tener una idea clara de la forma en que cada una participa dentro del sistema Ajax.

2.3.1. JavaScript

Es un lenguaje de programación interpretado, esto significa que existe en el navegador un intérprete que recibe las instrucciones en lenguaje JavaScript y una o muchas entradas para producir las salidas correspondientes. A diferencia de un compilador, que traduce las sentencias a un código objeto, el intérprete simplemente las ejecuta produciendo salidas.

Generalmente, el lenguaje JavaScript se utiliza para la creación de páginas dinámicas que contienen animaciones que la hacen más atractiva para el usuario, pero este lenguaje no se limita a esto, también es un elemento clave en Ajax, porque reúne a todos los demás elementos de Ajax para producir la

comunicación asíncrona que se describió anteriormente. En JavaScript se desarrollan un conjunto de subrutinas o procedimientos que realizan las actividades lógicas de Ajax.

Recientemente surgió la idea de utilizar JavaScript del lado del servidor, lo cual resultó ser una idea genial por las múltiples ventajas que presenta, más adelante, en el capítulo de Node.js, se desarrollará este tema a detalle.

2.3.2. JSON

El presente capítulo fue desarrollado porque se pueden obtener muy buenos resultados cuando se utiliza JSON en la programación con JavaScript. Regularmente, durante la programación web, llega el punto en el que se tienen que manipular grandes cantidades de información y lo ideal es que se haga de una manera sencilla y efectiva. La información tiene que estar organizada de tal manera que resulte fácil de comprender para el programador, en este punto resulta útil JSON.

JSON es el acrónimo de JavaScript *object notation*, es un formato estándar para el intercambio de datos. JSON surgió como una alternativa a XML y describe la información con una sintaxis específica, adquirió gran popularidad porque resulta muy fácil de utilizar cuando se programa con JavaScript. Una de sus cualidades es que puede ser leído por cualquier lenguaje programación, siempre que dicho lenguaje tenga una forma de procesar el texto con la sintaxis de JSON. Por lo anterior se dice que JSON encaja en Ajax junto a XML y XSLT, para el intercambio y manipulación de información.

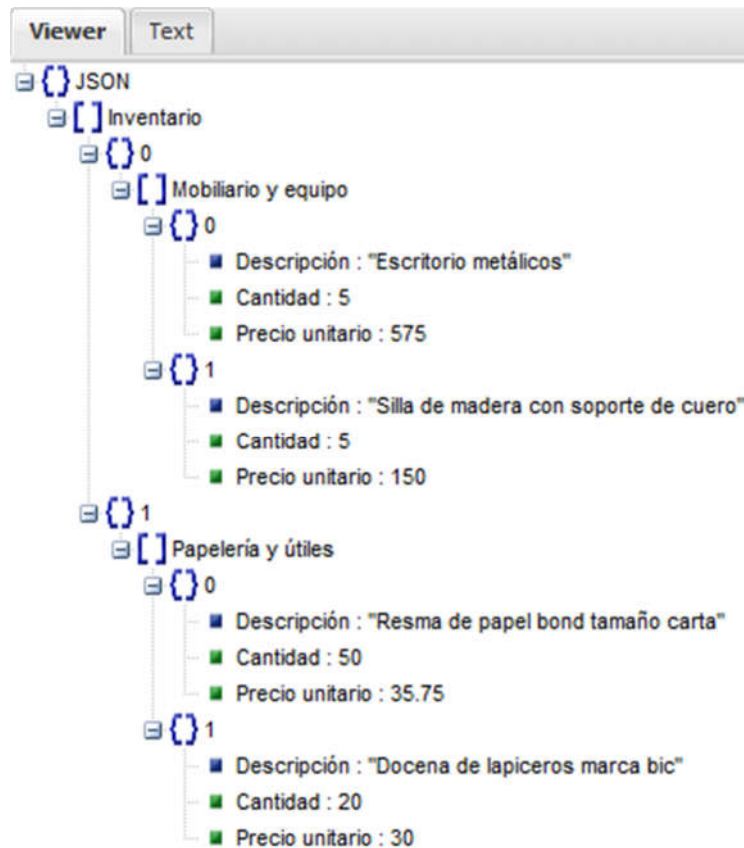
La sintaxis de JSON permite identificar fácilmente las relaciones jerárquicas presentes en la información. Para comprender mejor esta cualidad de JSON, se considerará el caso en el que cierto módulo web de una aplicación de contabilidad necesita manipular información sobre un inventario, además de manipularla debe intercambiarla. El código de JSON sería algo parecido a lo que se muestra en la figura 6, donde se observa fácilmente las relaciones jerárquicas de la información, es simple ver un elemento del inventario y saber a qué cuenta pertenece. En la figura 7 se observa un equivalente gráfico de la jerarquía mostrada en la figura 6.

Figura 6. Fracción de código en JSON con información de un inventario

```
1 {
2   "Inventario": [
3     {
4       "Mobiliario y equipo": [
5         {
6           "Descripción": "Escritorio metálicos",
7           "Cantidad": 5,
8           "Precio unitario": 575.00
9         },
10        {
11          "Descripción": "Silla de madera con soporte de cuero",
12          "Cantidad": 5,
13          "Precio unitario": 150.00
14        }
15      ]
16    },
17    {
18      "Papelería y útiles": [
19        {
20          "Descripción": "Resma de papel bond tamaño carta",
21          "Cantidad": 50,
22          "Precio unitario": 35.75
23        },
24        {
25          "Descripción": "Docena de lapiceros marca bic",
26          "Cantidad": 20,
27          "Precio unitario": 30.00
28        }
29      ]
30    }
31  ]
32 }
```

Fuente: elaboración propia, empleando Notepad Plus Plus v6.7.7.

Figura 7. **Equivalente gráfico del código mostrado en la figura 6**



Fuente: elaboración propia, empleando JSON Viewer.

2.3.3. XMLHttpRequest

Todas las aplicaciones que se realizan utilizando Ajax, deben realizar, como primer paso, una instancia del objeto XMLHttpRequest. Este objeto propio de JavaScript es muy importante porque permite que el cliente se comunique con el servidor en segundo plano, sin necesidad de recargar la página. Este objeto posee un conjunto de propiedades y métodos que se utilizan para establecer la comunicación.

Se debe considerar el navegador que se utilizará para ejecutar la aplicación, porque hay algunos navegadores antiguos, como Internet Explorer 6 y versiones anteriores a esta, que implementan el XMLHttpRequest como un objeto de tipo ActiveX, entonces, para estos navegadores debe instanciarse un ActiveXObject, de lo contrario no funcionará correctamente.

2.3.4. XML

XML es el acrónimo de *extensible markup language*, en español, lenguaje de marcas extensible. Este lenguaje es un estándar utilizado para la manipulación y el intercambio de información, surgió a partir de GML (*general markup language*) un lenguaje desarrollado por la empresa IBM en los años setenta para almacenar grandes cantidades de información sin que esta perdiera su estructura o su sentido.

Este lenguaje fue desarrollado por la *world wide web consortium* (W3C), desde que lo crearon han continuado con su desarrollo. Es un estándar ampliamente aceptado y utilizado en el desarrollo de aplicaciones web, principalmente desde que se adaptó el lenguaje XML a HTML. Como se mencionó anteriormente, JSON surgió como una alternativa a XML, lo significa que la información descrita con XML también puede ser descrita con JSON, que también es utilizado para la manipulación y el intercambio de información. En la figura 8 se muestra el código XML equivalente al código JSON mostrado en la figura 6.

Figura 8. **Código XML equivalente al código JSON mostrado en la figura 6**

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <Inventario>
3   <Mobiliario_y_equipo>
4     <Elemento>
5       <Descripción>Escritorio Metálicos</Descripción>
6       <Cantidad>5</Cantidad>
7       <Precio_unitario>575.00</Precio_unitario>
8     </Elemento>
9     <Elemento>
10      <Descripción>Silla de madera con soporte de cuero</Descripción>
11      <Cantidad>5</Cantidad>
12      <Precio_unitario>150.00</Precio_unitario>
13    </Elemento>
14  </Mobiliario_y_equipo>
15  <Papelería_y_útiles>
16    <Elemento>
17      <Descripción>Resma de papel bond tamaño carta</Descripción>
18      <Cantidad>50</Cantidad>
19      <Precio_unitario>35.75</Precio_unitario>
20    </Elemento>
21    <Elemento>
22      <Descripción>Docena de lapiceros marca bic</Descripción>
23      <Cantidad>20</Cantidad>
24      <Precio_unitario>30.00</Precio_unitario>
25    </Elemento>
26  </Papelería_y_útiles>
27 </Inventario>
```

Fuente: elaboración propia, empleando NetBeans 8.0.2.

2.3.5. XSLT

XSLT es el acrónimo de *extensible stylesheet language transformations*, es un lenguaje estándar desarrollado por la W3C que permite transformar documentos XML a otros documentos XML con estructura distinta o a documentos con otro formato distinto del XML.

Este lenguaje es muy útil para diversos casos, pero si se considera una situación en la que se tiene una gran cantidad de información almacenada en un archivo formato XML y conviene mantenerla en este formato porque las aplicaciones manipulan esta información constantemente, pero, además se necesita que los usuarios puedan ver esta información en una tabla dentro de una página web. Se podría fácilmente definir un XSLT con reglas de transformación que produzcan un texto en formato HTML que contenga una tabla que pueda mostrarse en el navegador para los usuarios. El código de XSLT que realiza esta transformación es relativamente corto y es mucho más fácil hacer esto y no crear una página HTML a partir del XML con algún otro lenguaje.

2.3.6. DOM

En el lenguaje XML se puede escribir información fácilmente, pero procesarlo no es una tarea tan sencilla. Es por ello que en los diferentes lenguajes existen librerías que permiten procesar este lenguaje para manipularlo fácilmente. DOM se utiliza para esto y está formado por un conjunto de herramientas que permiten manipular documentos XML, esto significa que con DOM también se puede manipular documentos HTML y XHTML, que también son lenguajes de marcas.

DOM es el acrónimo de *document object model* y es una herramienta muy útil en el desarrollo de aplicaciones web. Lo primero que hace DOM al analizar un texto XML es generar una estructura de datos, esta estructura es un árbol que modela la información del XML respetando las relaciones jerárquicas que dicha información contiene. Tomando en consideración el código XHTML que se muestra en la figura 9, esta información contiene relaciones jerárquicas que DOM considera cuando ejecuta sus acciones, por ejemplo, dentro del segmento

BODY se encuentran dos segmentos DIV, que en la estructura de datos generada por DOM deberían ser sus descendientes.

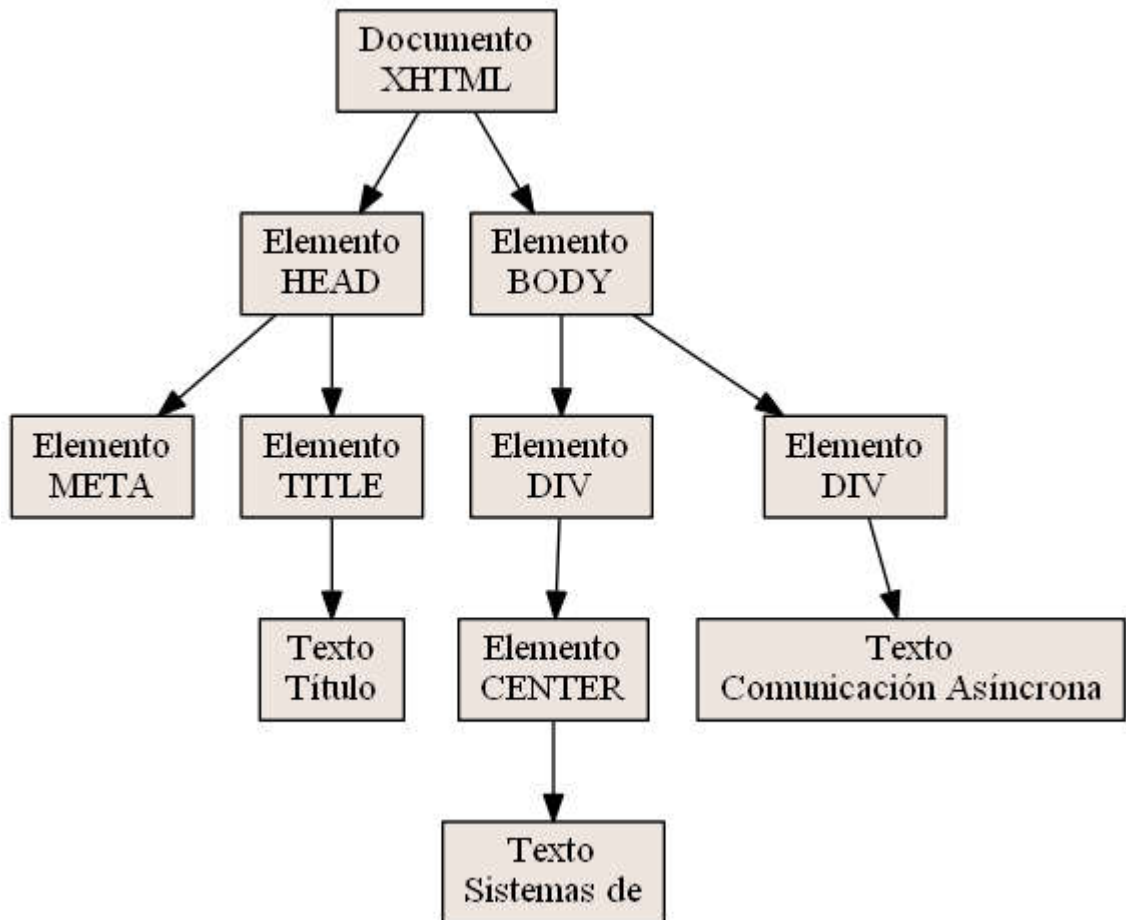
Figura 9. Código XHTML ejemplo

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="Content-Type" content="text/html;
      charset=iso-8859-1" />
    <title>T&iacute;tulo</title>
  </head>
  <body>
    <div>
      <center>Sistemas de </center>
    </div>
    <div>Comunicaci&oacute;n As&iacute;nrona</div>
  </body>
</html>
```

Fuente: elaboración propia, empleando NetBeans 8.0.2.

Como se mencionó anteriormente, la primera tarea que DOM ejecuta es crear una estructura de datos que contiene la información del texto analizado, esta estructura puede manipularse fácilmente eliminando, modificando o agregando nodos. En la figura 10 se muestra el árbol que DOM generaría para el código mostrado en la figura 9.

Figura 10. Ejemplo de estructura de datos generada por DOM



Fuente: elaboración propia, empleando Graphviz.

2.3.7. XHTML

El lenguaje XHTML surge como una adaptación del lenguaje HTML al lenguaje XML. Muchas de las sentencias que son válidas en lenguaje HTML no lo son en XHTML, principalmente porque el estándar aplicado es más rígido. Esto tiene muchas ventajas, pero puede resultar molesto para los programadores que están acostumbrados a utilizar HTML. Dentro de los

elementos descritos por Garret para Ajax, se encuentra XHTML porque busca crear una presentación basada en estándares, esto no significa que no pueda utilizarse HTML junto a Ajax.

Se ha debatido mucho sobre si HTML es superior a XHTML para el desarrollo de aplicaciones web o viceversa, y no hay una respuesta definitiva porque las opiniones están muy divididas. Sin embargo, la gran cantidad de nuevas opciones que brinda HTML en su versión cinco seguramente harán que los desarrolladores se inclinen a utilizar HTML en sus aplicaciones web.

2.3.8. CSS

CSS son las siglas de *cascading style sheet*, las hojas de estilo fueron desarrolladas con el objetivo de definir un sistema que permitiera aplicar estilos consistentes a los documentos electrónicos. Conforme la web evoluciona, se vuelve más importante la presentación y el diseño de las páginas, porque en las páginas web muchas empresas han encontrado una efectiva manera de vender productos y servicios y las páginas que resultan atractivas al usuario venden.

Actualmente, CSS va por su versión tres y cuenta con muchas opciones muy flexibles que facilitan el desarrollo del diseño de la página. Los resultados que los diseñadores pueden obtener utilizando CSS tres son muy superiores a los que se obtenían con sus versiones anteriores.

2.4. Modelo de eventos

En programación de aplicaciones web, el modelo de eventos consiste en que los *scripts* permanecen en un estado de espera hasta que sucede algún evento que dispara o activa una subrutina que ejecuta un conjunto de

instrucciones, luego de ejecutar dichas instrucciones el *script* vuelve a un estado de espera y permanece así hasta que suceda un nuevo evento.

El modelo de eventos se introdujo en la versión cuatro de HTML y forma parte del nivel más básico de DOM. Cada elemento de HTML tiene un conjunto de posibles eventos a los que puede estar asociado, de igual manera, un evento puede estar asociado a muchos elementos diferentes, existe una amplia variedad de eventos que se puede utilizar para desarrollar las tareas dentro de la página.

2.4.1. Flujo de eventos

Un mismo evento puede disparar muchas acciones realizadas por diferentes elementos. Por ejemplo, si se considera el caso en el que dentro de una página se encuentra un DIV, dentro de dicho DIV se encuentra un botón y el botón tiene un conjunto de acciones asociadas al evento clic; el DIV tiene otras acciones distintas asociadas a ese mismo evento y la página posee otras acciones asociadas también al evento clic. Cuando se produzca el evento clic sobre el botón, se ejecutarán las acciones del botón, las del DIV y las de la página en cierto orden. Este orden en el que se ejecutan las acciones para el evento constituye el flujo de eventos.

El orden en el que se ejecutan las acciones depende del modelo de flujo de eventos que el navegador utilice. Mozilla Firefox, por ejemplo, utiliza el modelo de propagación de eventos que ejecuta las acciones del elemento más específico asociado a dicho evento al más general. Según este modelo, en el ejemplo anterior deberían ejecutarse primero las acciones del botón, luego las del DIV y por último las de la página. Existe también otro modelo de flujo de eventos conocido como modelo de captura de eventos, que funciona de manera

opuesta al anterior, este ejecuta las acciones del elemento más general al elemento más específico.

2.4.2. Manejadores de eventos

En la programación orientada a eventos, las aplicaciones esperan hasta que un evento ocurra para realizar ciertas acciones, estas acciones están desarrolladas especialmente para dicho evento. Al conjunto de instrucciones que se ejecutan cuando sucede cierto evento se le da el nombre de manejadores de eventos. Cuando se definen funciones específicas que se ejecutarán como respuesta a los eventos, se les da el nombre de funciones manejadoras.

La forma más sencilla de definir un manejador es colocándolo como un atributo en la etiqueta de HTML que corresponde al elemento que estará asociado al evento. Otra opción es establecer una función externa que se llama desde un atributo en la etiqueta del elemento que será asociado al evento. La última y mejor opción es separar la programación en JavaScript del código HTML para tener una página limpia, esto se hace mediante las propiedades DOM de los elementos HTML.

2.5. Marcos de trabajo y librerías complementarias

El uso de marcos de trabajo y librerías facilita mucho la labor de los desarrolladores. La mayoría de estas librerías funcionan en los navegadores más populares, por lo que las preocupaciones por el tema de la compatibilidad son mínimas. Además, es muy importante que el desarrollador conozca todas estas herramientas, o por lo menos las más populares, para evitarse la tarea innecesaria de desarrollar una herramienta que ya existe. A continuación se

describen algunas de las librerías y marcos de trabajo más populares para JavaScript y Ajax.

2.5.1. Prototype

Es un marco de trabajo que se utiliza para desarrollar aplicaciones web con JavaScript y Ajax. Su creador es Sam Stephenson, quien en las últimas versiones de Prototype ha contado con la colaboración de otros programadores. Desde su creación ha sido muy importante, porque ofrece muchas opciones que facilitan considerablemente el desarrollo de aplicaciones web con Ajax, además, sirvió como base para muchas otras librerías. Las últimas versiones de este marco de trabajo cuentan con una buena documentación de todos los métodos y funciones que lo componen. Esta documentación cuenta con la descripción de métodos y atributos y con algunos ejemplos básicos pero muy útiles.

Tiene muchas funciones equivalentes a las funciones de JavaScript para DOM, pero están considerablemente reducidas, al utilizar Prototype se obtiene código es más claro y ordenado. Cuenta con funciones para la manipulación de formularios, manejo de estructura de datos, manejo de elementos enumerables y, lo más importante, cuenta con una serie de métodos para el uso de Ajax.

2.5.2. jQuery

jQuery es una librería de JavaScript creada por John Resig, pero hay muchos programadores que contribuyen actualmente para su desarrollo. Muchos de los usuarios de Prototype lo cambiaron por jQuery.

Esta librería facilita la selección de elementos dentro de la estructura DOM de la página, cuenta con opciones para la creación de funciones para eventos, efectos visuales y CSS. Es posible detectar el navegador en el que la aplicación se está ejecutando de manera sencilla y, lo que interesa, cuenta funciones que facilitan el uso de Ajax.

2.5.3. Mootools

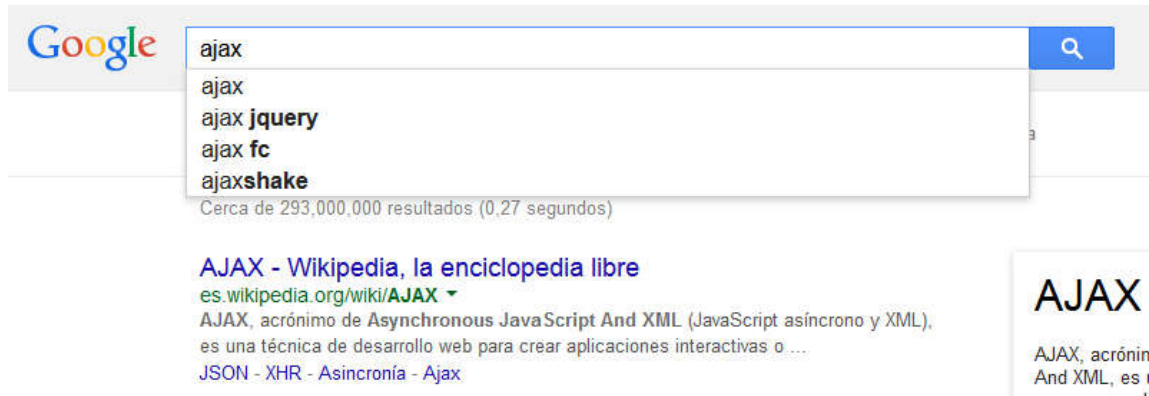
Esta librería, al igual que las anteriormente descritas, provee un conjunto de funciones que facilitan el desarrollo de aplicaciones web con JavaScript. Tiene la característica especial de que cuando se descarga es posible seleccionar únicamente los componentes que se desean utilizar, esto significa que la librería solo contiene lo necesario y por lo tanto es sumamente ligera.

2.6. ¿Quién está utilizando Ajax?

Ajax es utilizado principalmente en sitios en los que es muy importante una interacción fluida y rápida con el usuario, de no ser así, el uso de estas aplicaciones web resultaría tedioso y desagradable.

Por ejemplo, si un servicio como Google Maps no tuviera Ajax, sería bastante tedioso utilizarlo porque cada vez que el cliente necesite ir al servidor en busca de información para repintar el mapa, tendría que recargar la página, esto resultaría tardado y no se podría navegar en el mapa con la fluidez que se desea. Google siempre ha demostrado gran interés en la comodidad y la satisfacción del usuario, Ajax es una herramienta importante para lograr esto. Google ha hecho uso de Ajax en sus productos más populares, como Gmail, Google Groups, Google Suggest, Google Maps, entre otros. En la figura 11 se muestra una imagen de Google Suggest ejecutándose.

Figura 11. Ejemplo de Google Suggest ejecutándose



Fuente: elaboración propia, empleando Google Suggest.

2.7. El papel de Ajax en la web en tiempo real

La web en tiempo real realiza acciones en lapsos muy cortos, porque está sujeta a restricciones temporales muy estrictas, ya que tiene como principal objetivo poner la información a disposición del usuario inmediatamente después de que se genera.

Considerando una situación en la que existe cierto sistema desarrollado en un entorno web que constantemente está sufriendo cambios por las acciones de múltiples usuarios; todos los usuarios que hacen modificaciones desean saber los resultados de sus propias modificaciones en el sistema y los resultados de las modificaciones de otros usuarios. Para que todos los usuario conozcan la información nueva justo después de que cambia, hace falta que la comunicación entre el cliente y el servidor sea muy rápida. Está claro que el tener que recargar la página representa un consumo innecesario de tiempo, Ajax es una opción viable para resolver este problema, por supuesto que hay que considerar muchos otros factores para lograr un óptimo desempeño del

sistema pero, el hecho de incorporar un sistema de comunicación asíncrona por medio de Ajax ya mejora por mucho la rapidez de la interacción entre el cliente y el servidor.

3. NODE.JS, UN INTÉRPRETE ASÍNCRONO PARA JAVASCRIPT DEL LADO DEL SERVIDOR

JavaScript, desde su creación, ha sido utilizado para programar acciones del lado del cliente, es decir, acciones que serán ejecutadas por un intérprete del navegador. JavaScript es un lenguaje bastante flexible que generalmente es utilizado para hacer el entorno web un poco más dinámico para el usuario.

JavaScript puede utilizarse del lado del cliente, por ejemplo, para crear pestañas que cambian de forma cuando el cursor está sobre ellas o gráficas que varían a lo largo del tiempo porque modelan información cambiante, también puede ser utilizado para implantar funciones Ajax, como se mostró en el capítulo anterior.

El utilizar JavaScript del lado del servidor es una idea innovadora que probablemente no se le había ocurrido a nadie, porque los programadores tenían implantada en su mente la idea de que JavaScript tenía que usarse para programar acciones en el cliente. La idea de utilizar JavaScript en el servidor surge en el 2009, cuando Ryan Dahl desarrolla Node.js.

3.1. ¿Qué es Node.js?

Node.js es un intérprete asíncrono para JavaScript del lado del servidor. El principal objetivo de Node.js, según su página web oficial, es brindar una forma fácil y rápida de construir aplicaciones de red escalables. Este objetivo se alcanza por la capacidad de los servidores Node.js para aceptar una gran

cantidad de conexiones simultáneamente con una carga mínima, gestionándolas a través de un solo proceso.

El utilizar JavaScript, un lenguaje interpretado, en un motor tan poderoso como lo es V8, hace que la ejecución de las tareas programadas sea increíblemente rápida. Si tenemos JavaScript, tanto en el cliente como en el servidor, la interacción entre ambos es bastante sencilla, principalmente si consideramos que el intercambio de información puede hacerse mediante JSON.

Existen muchos casos exitosos de sistemas que han implementado Node.js, quizás uno de los más populares es el de LinkedIn, un sitio web orientado a los negocios y al contacto entre profesionales que posee una lógica similar a la de una red social. Luego de migrar a Node.js, LinkedIn redujo el número de sus servidores de treinta a solamente tres e incrementó la velocidad de ejecución hasta veinte veces en algunos escenarios³.

Considerando una situación en la que se tiene un servidor con Apache instalado, si llega el punto en el que el servidor ya no tiene la capacidad de atender las solicitudes de los clientes porque son demasiadas, se debe incrementar la capacidad del sistema agregando más servidores. Si el número de clientes que debe atender el sistema vuelve a incrementar, entonces es necesario agregar más servidores, esto representa muchos costos, mientras más servidores se agreguen, mayor será el costo. El principal problema para el caso anterior es que el número de clientes que puede atender un servidor es bastante limitado.

³ *LinkedIn moved from rails to node: 27 servers cut and up to 20x faster.* <http://highscalability.com/blog/2012/10/4/linkedin-moved-from-rails-to-node-27-servers-cut-and-up-to-2.html>. Consulta: abril de 2015.

Node.js puede atender gran cantidad de solicitudes, principalmente porque trabaja utilizando un solo hilo para gestionar todas las conexiones, esto significa que si aumentara el número de clientes, habría que agregar menos servidores que los que se agregarían si se utilizara Apache. Entonces, se puede decir que el sistema con Node.js es más escalable que el sistema con Apache, porque incrementar su capacidad requiere un menor esfuerzo y representa un menor costo.

La velocidad de Node.js lo hace ideal para implementar aplicaciones con restricciones temporales, además, cuenta con un poderoso módulo que permite establecer comunicación asíncrona con el cliente, este módulo es Socket.IO y se ajusta bien a las necesidades de la web en tiempo real.

3.2. ¿Cómo funciona Node.js?

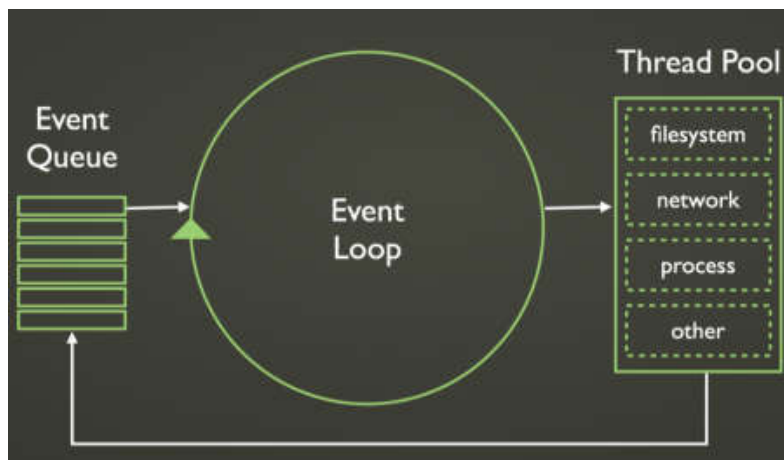
Node.js no consiste solamente en utilizar un intérprete para ejecutar código de JavaScript del lado del servidor, su funcionamiento es más complejo. Node.js usa un motor basado en el motor V8 de Google para interpretar el código de JavaScript. El motor V8 fue diseñado para ejecutar JavaScript en el navegador y se caracteriza por ser muy rápido, el motor de Node.js también lo es.

En los servidores web como Apache, se genera un hilo del sistema operativo para cada conexión que se establece con un cliente, esto es básicamente lo que limita el número de conexiones que un servidor puede aceptar simultáneamente. A cada hilo se le asigna un espacio de memoria determinado y cuando la memoria llega a su límite, el servidor ya no puede atender más solicitudes. Por otro lado, en Node.js se tiene un solo hilo para todas las conexiones y cada vez que un cliente se conecta, en lugar de crear un

nuevo hilo, se dispara un evento. Cada vez que el cliente envía datos se dispara un evento y así todas las acciones se realizan a través de eventos.

Además de que el motor V8 ejecuta JavaScript a altas velocidades, la otra razón por la que Node.js es tan rápido es el enfoque asíncrono, el cual depende directamente de algo llamado bucle de eventos. El hilo principal agrega en una cola todos los eventos recibidos, el bucle de eventos está siempre pendiente de los eventos que puedan suceder, cuando detecta alguno en la cola de eventos lo ingresa en el *thread pool*, en donde las acciones asociadas a los múltiples eventos se realizan simultáneamente de manera asíncrona. Esto significa que el bucle de eventos delega las tareas al *thread pool* e inmediatamente después de hacerlo está listo para detectar un nuevo evento. Es muy importante que no se programen funciones síncronas dentro de Node.js, porque si esto se hiciera, el bucle de eventos se quedaría esperando y como únicamente se tiene un hilo, no se podría recibir ningún otro evento hasta que se haya terminado de ejecutar el anterior. Esta dinámica del bucle de eventos se observa en la figura 12.

Figura 12. **Gráfica del bucle de eventos de Node.js**



Fuente: *What is Node.js*. <http://thejackalofjavascript.com/nodejs/>. Consulta: abril de 2015.

3.3. El motor V8 de JavaScript

V8 es el motor de JavaScript de alto rendimiento creado por Google, es de código abierto y está escrito en C++, es utilizado en el navegador Google Chrome, el navegador de código abierto de Google. Este motor posee la característica de poder incrustarse en cualquier aplicación de C++. El motor V8 fue adaptado para utilizarse en Node.js y se le realizaron múltiples cambios para realizar únicamente las tareas que eran necesarias para programar del lado del servidor. Por ejemplo, el V8 utilizado en Node.js no tiene las funciones DOM, ya que no tiene sentido que estas funciones existan del lado del servidor.

3.4. NPM

NPM puede referirse a un *node package module* o puede referirse al *node package manager*, el hecho de que se tenga un mismo acrónimo para ambos elementos de Node.js puede prestarse a confusiones.

El *node package manager* es el administrador de paquetes que utiliza Node.js. Es uno de los elementos más importantes cuando se desarrolla con Node.js, porque con ayuda de él se importan todas las librerías o módulos que se necesitan para desarrollar aplicaciones, es bastante sencillo y se hace ejecutando cierta instrucción desde consola.

Un *node package module* es cualquier modulo que pueda utilizarse para programar en Node.js, es decir, lo que se importa con ayuda del *node package manager* es un *node package module*, o muchos, según las necesidades. Un ejemplo de *node package module* es Socket.IO, una librería que puede ser utilizada para establecer comunicación asíncrona entre el cliente y el servidor.

3.5. Marco de trabajo monohilo, asíncrono y dirigido por eventos

Como se mencionó anteriormente, Node.js trabaja con un solo hilo sin importar el número de conexiones que reciba, porque detecta cada conexión como un evento. Cuando este evento se encuentra en la cola de eventos y es detectado por el bucle de eventos, entonces este bucle lo envía al *thread pool*, en donde son ejecutadas las acciones relacionadas a dicho evento. Para que el funcionamiento con un solo hilo de Node.js sea correcto, es necesario que no exista código bloqueante, es decir, es necesario que todas las tareas sean tratadas de manera asíncrona.

Bajo este contexto, el término asíncrono se aplica a ejecución tareas. Si se tiene un conjunto de tareas que deben ejecutarse de manera síncrona, se ejecutará una tarea primero, cuando esta tarea termine su ejecución, se empezará a ejecutar la segunda tarea, cuando la segunda termine su ejecución, seguirá la tercera y así sucesivamente hasta terminar de ejecutarlas todos. Si se tiene un conjunto de tareas que deben ejecutarse de manera asíncrona, se empezará a ejecutar la primera tarea e inmediatamente después, sin que la primer tarea se haya terminado, empezará a ejecutarse la segunda tarea, inmediatamente después empezará a ejecutarse la tercera y así sucesivamente hasta terminar de ejecutar todas las tareas. Esto significa que no hace falta esperar a que la primer tarea termine para empezar con la segunda o esperar a que la segunda termine para empezar con la tercera.

Con la anterior explicación queda claro que la ejecución síncrona de tareas tiene la ventaja de que existe un control completo del orden en el que las tareas se ejecutan. En las tareas asíncronas no se tiene este nivel de control porque las tareas prácticamente se empiezan a ejecutar al mismo tiempo, ya que una empieza su ejecución inmediatamente después de la otra y se ejecutan

de forma paralela, es decir, al mismo tiempo. Si existiera dependencia entre tareas, manejar esta dependencia sería un poco más complicado que con las tareas síncronas, porque se tendría que saber con certeza la duración de las tareas, o bien, manejar retardos en la ejecución de mismas. En Node.js no se sabe cuándo ni en qué orden se ejecutarán las tareas porque estas deben ejecutarse asíncronamente, pero puede manipularse el orden utilizando la función `process.nextTick` para retrasar la ejecución dentro de la cola de eventos.

Por otro lado, el tiempo de ejecución de las tareas asíncronas es muy inferior al tiempo de ejecución de las tareas síncronas. Por ejemplo, considerando un sistema en el que se deben ejecutar diez tareas y el tiempo que se requiere para ejecutar cada tarea es de dos segundos, además, se sabe que inmediatamente después de iniciar la ejecución de una tarea el sistema está listo para iniciar la ejecución de una nueva tarea, por lo que el tiempo entre el arranque de una tarea y el arranque de otra tarea es despreciable.

Si en este sistema se quisiera ejecutar las tareas de manera síncrona, la duración de dicha ejecución sería de veinte segundos porque tiene que ejecutarse la primera y hasta que se termine la primera puede iniciar la segunda y así sucesivamente hasta terminar. Pero, si se ejecutaran las tareas de manera asíncrona, el tiempo de ejecución sería de dos segundos porque podrían ejecutarse las diez tareas simultáneamente y aunque unas tareas arrancaran después que otras, se sabe que el tiempo entre el arranque de una tarea y otra es tan pequeño que es despreciable, por lo que la duración total sería de dos segundos.

Considerando lo anterior, se tiene que la mejor opción para las aplicaciones de la web en tiempo real es la asíncrona, porque es muy

importante el tema del tiempo. Node.js utiliza justamente un marco de trabajo asíncrono, con un solo hilo y dirigido por eventos. El modelo dirigido por eventos de Node.js posee el mismo funcionamiento que fue descrito en el capítulo anterior de Ajax.

3.6. El modelo no bloqueante de entrada y salida

Considerando que Node.js posee un solo hilo de ejecución, es muy importante que cuando se utilice, todo se desarrolle con un modelo no bloqueante, es decir, todo lo que se programe debe ser asíncrono porque si se ejecuta una tarea síncrona que tenga que esperar, todo el sistema Node.js estará esperando. Si se tiene código bloqueante, todo el servidor estará bloqueado y esto implica no responder a nuevas solicitudes. Además, se tienen que capturar los errores inesperados, porque si estos errores implican una espera, todo el sistema se verá afectado por dicha espera.

También es importante manejar eventos de *timeout*, que estarán directamente ligados a las operaciones de comunicación. Por ejemplo, si cierto *socket* espera la conexión de un cliente, pero esta no llega luego de cierto tiempo, se dispara un evento de *timeout* en el que se indica que el tiempo de espera ha vencido y que el *socket* dejará de esperar una conexión.

3.7. Módulo Socket.IO y Websockets

Los Websockets permiten establecer comunicación asíncrona entre el cliente y el servidor, este tipo de comunicación es bidireccional y puede iniciarse tanto en el cliente como en el servidor. Los Websockets se inician con un *handshake* HTTP, es decir, una negociación para establecer la comunicación, tanto el cliente como el servidor deben soportar esta negociación. En este tipo

de *sockets* solo es posible transmitir texto, a diferencia del TCP convencional en el que era posible transmitir *streams* de bytes. Los Websockets son ideales para la web en tiempo real, porque facilitan establecer comunicaciones asíncronas en las que el servidor puede enviar información al cliente cuando lo desee, específicamente cuando esta información se genera.

Socket.IO es un módulo en JavaScript para Node.js que facilita considerablemente el uso de Websockets, es posible instalarlo con ayuda de NPM y es bastante sencillo utilizarlo del lado del cliente. Cabe mencionar que la gran desventaja de estos *sockets* es que no soportan la interacción con clientes que utilicen Websockets estándar, sin embargo, es bastante sencillo implementar el Socket.IO del lado del cliente, por lo que esta desventaja resulta irrelevante en la mayoría de los casos. Una vez conectado el cliente y el servidor a través del *socket*, puede establecerse la comunicación y todos los mensajes que se envían son texto, el cliente conectado al servidor hasta que decida cerrar el *socket* para cortar la comunicación.

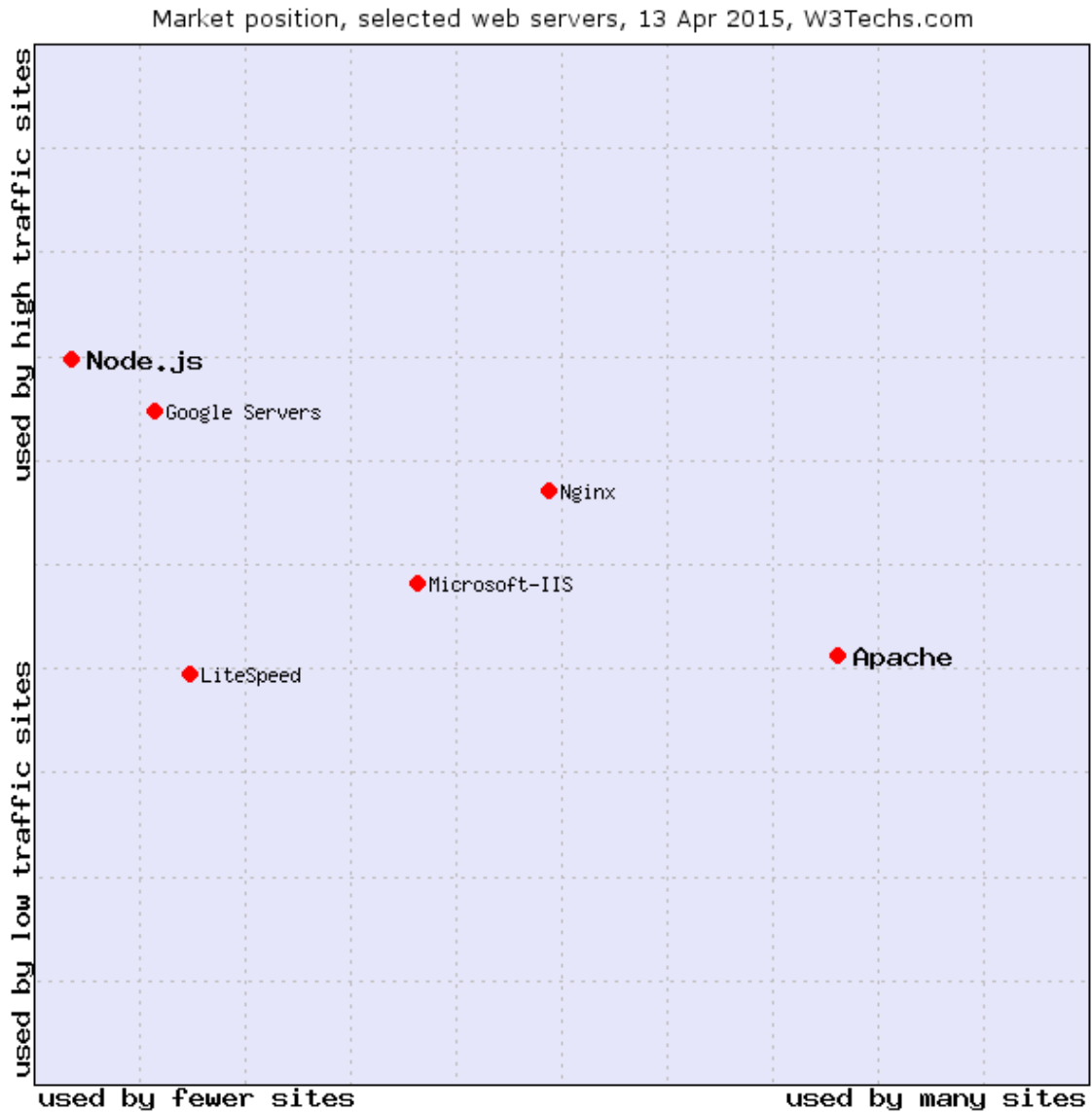
3.8. El papel de Node.js en la web en tiempo real

Node.js es de suma importancia en la web de tiempo real porque además de ejecutar tareas a gran velocidad, permite un alto nivel de concurrencia. Es bastante grande la cantidad de conexiones que Node.js puede manejar en comparación con otros servidores web que creaban un hilo para cada conexión. En la actualidad, esto es especialmente importante porque cada vez es más grande el número de personas que se inclinan a utilizar aplicaciones web, principalmente por la creciente popularidad de dispositivos móviles y las redes sociales. Esto significa que los servidores web tienen que tener la capacidad de soportar grandes cantidades de solicitudes simultáneamente. En respuesta a ello, algunas empresas han decidido migrar a esta nueva tecnología llamada

Node.js, que aunque es joven ha demostrado que puede dar muy buenos resultados si sabe en qué momento es adecuado utilizarla.

Para describir más fácilmente la posición en el mercado de Node.js, se muestra en la figura 13 una gráfica realizada por W3Techs. Esta gráfica muestra la posición en el mercado de diferentes servidores web, considerando la popularidad y el tráfico de cada uno, como se puede observar, Node.js es usado por pocos sitios pero estos sitios tienen cantidades muy altas de tráfico, esta página se considera bastante confiable porque se dedica a las encuestas de tecnologías web.

Figura 13. **Posición en el mercado de diferentes servidores web**



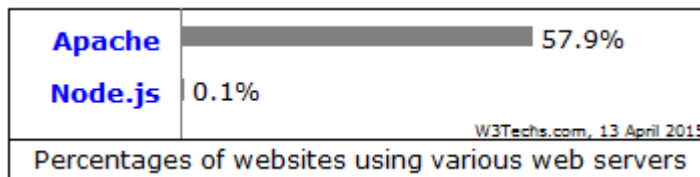
Fuente: *Comparison of the usage of Apache vs. Node.js for websites.*

<http://w3techs.com/technologies/comparison/ws-apache,ws-nodejs>. Consulta: abril de 2015.

Cabe mencionar que Node.js no es un servidor web tan popular como lo es Apache u otros servidores web con más tiempo de existencia. En la figura 14 se observa una gráfica que muestra el porcentaje de sitios web que utilizan el

servidor web seleccionado, esta gráfica toma en cuenta todos los sitios conocidos por W3Techs.

Figura 14. **Popularidad de Node.js versus Apache**



Fuente: *Comparison of the usage of Apache vs. Node.js for websites.*

<http://w3techs.com/technologies/comparison/ws-apache,ws-nodejs>. Consulta: abril de 2015.

A lo anterior se debe agregar que la cantidad de información que los servidores deben manejar es enorme y la web en tiempo real necesita que esta información sea manipulada de la manera más rápida que sea posible, por lo que vale la pena tomar en cuenta todas las opciones en gestores de bases de datos, principalmente los NOSQL que brindan muy buenas opciones en lo que se refiere a rendimiento.

Como se pudo observar en la gráfica de la figura 14, que muestra la uso de Apache y Node.js en los sitios conocidos por W3Techs, Node.js no es tan popular. Sin embargo, vale la pena tomarlo en cuenta por todas las características que anteriormente se mencionaron, sin lugar a duda dentro de algunos años Node.js va a ocupar un lugar importante en el desarrollo de aplicaciones web, principalmente en la web en tiempo real.

3.9. Pruebas de rendimiento, Node.js versus Apache

Para dar una idea más clara de la diferencia de rendimiento entre Node.js y Apache con PHP, se realizaron pruebas utilizando el comando `ab` de las utilidades de Apache, `ab` (ApacheBench) es la herramienta de evaluación comparativa del servidor Apache HTTP. Este comando recibe múltiples parámetros, pero para efectos de esta prueba se indicará solamente el número de peticiones, el nivel de concurrencia, la URL a la que se harán las peticiones y se le indicará que no se detenga al detectar algún error, sino continúe con el envío de peticiones. Las especificaciones de la máquina utilizada para realizar las pruebas se observan en la tabla II.

Tabla II. **Especificaciones de software y hardware de la computadora utilizada en las pruebas de rendimiento**

Característica	Soporte
Memoria RAM	2 GB
Disco duro	500 GB
Procesador	Celeron (R) Dual-Core T3300 (2.00 GHz)
Sistema operativo	Ubuntu 14.04 LTS
Versión Apache	2.4.7
Versión PHP	5.5.9
Versión Node.js	V0.10.25

Fuente: elaboración propia.

Para las pruebas se programó una aplicación en PHP que mostraba en el navegador un mensaje simple, se observa el código de dicha aplicación en la figura 15.

Figura 15. **Código PHP utilizado en pruebas de rendimiento**

```
1  <?php
2  echo '<p>Hola mundo!!!</p>';
```

Fuente: elaboración propia, empleando NetBeans.

También se programó una aplicación de Node.js equivalente que mostraba en el navegador el mismo mensaje, se observa el código de dicha aplicación en la figura 16.

Figura 16. **Código Node.js utilizado en pruebas de rendimiento**

```
1  var sys = require('sys'),
2  http = require('http');
3
4  http.createServer(function(request, response) {
5    response.writeHead(200, {'Content-Type': 'text/html'});
6    response.write('<p>Hola mundo!!!</p>');
7    response.end();
8  }).listen(8000);
```

Fuente: elaboración propia, empleando NetBeans.

Para las pruebas, en ambos casos, se realizaron un total de cien mil peticiones con un nivel de concurrencia de quinientos clientes. Los resultados para Apache se muestran en la tabla III y los resultados para Node.js se muestran en la tabla IV.

Tabla III. **Resultados de las pruebas de rendimiento de Apache**

Elemento de la prueba	Resultado
Nivel de concurrencia	500 clientes
Tiempo tomado para las pruebas	52,422 s
Peticiones completas	100 000 peticiones
Peticiones fallidas	0 peticiones
Total transferido	20 700 000 B
HTML transferido	2 000 000 B
Peticiones por segundo	1907,59 [#s] (media)
Tiempo por petición	262,110 [ms] (media)
Tiempo por petición	0,524 [ms] (media, a través de todas las peticiones concurrentes)
Velocidad de transferencia	385,62 [KB/s] recibidos

Fuente: elaboración propia.

Tabla IV. **Resultados de las pruebas de rendimiento de Node.js**

Elemento de la prueba	Resultado
Nivel de concurrencia	500 clientes
Tiempo tomado para las pruebas	37,149 s
Peticiones completas	100 000 peticiones
Peticiones fallidas	0 peticiones
Total transferido	12 000 000 B
HTML transferido	2 000 000 B
Peticiones por segundo	2 691,86 [#s] (media)
Tiempo por petición	185,745 [ms] (media)
Tiempo por petición	0,371 [ms] (media, a través de todas las peticiones concurrentes)
Velocidad de transferencia	315,45 [KB/s] recibidos

Fuente: elaboración propia.

Como se observó en los resultados, el rendimiento obtenido con Node.js es bastante superior. Para los resultados en el entorno de prueba se tiene que,

el tiempo total de respuesta de Node.js es menor que el tiempo total de respuesta de Apache en un 29,13 %.

4. CASO DE ESTUDIO

En este capítulo se analiza un sistema de subastas en línea requerido por una empresa que se dedica a las subastas de vehículos. Utilizando las tecnologías descritas en capítulos anteriores y el modelo de la web en tiempo real, se propone un diseño que cubre los requerimientos de dicha empresa.

El principal objetivo de este capítulo es brindar un ejemplo práctico del uso de sistemas de comunicación asíncrona en la web en tiempo real.

4.1. Descripción del caso de estudio

Cierta empresa dedicada a las subastas de vehículos desea por primera vez incorporar a sus servicios un sistema de subastas en línea. Solicita que la aplicación sea desarrollada en un entorno web, de tal manera que cualquiera de sus clientes registrados previamente en el sistema pueda participar en sus subastas; accediendo a la aplicación desde su computadora o dispositivo móvil.

La empresa desea que los participantes de las subastas conozcan las ofertas de los otros participantes de manera inmediata, para que todos tengan las mismas oportunidades dentro de la subasta. Para ello, es necesario implementar un sistema que muestre la información de todas las ofertas en tiempo real.

Actualmente, esta empresa posee un sistema que gestiona la información de sus clientes, empleados, subastas y vehículos, dicho sistema fue creado para el uso interno de la empresa y debe ampliarse para incorporar las

subastas en línea. El sistema actualmente posee un servidor con las características mostradas en la tabla V, utiliza como servidor web Apache 2.4.7 con PHP 5.5.9 y como DBMS MySQL 5.5.43. Este mismo servidor se utilizará para proveer las funciones del nuevo módulo de subastas.

Tabla V. **Especificaciones de software y hardware del servidor utilizado por la empresa de subastas**

Característica	Soporte
Memoria RAM	4 GB
Disco duro	500 GB
Procesador	Intel Core i5 (2.53 GHz)
Sistema operativo	Ubuntu 14.04 LTS Server
Versión Apache	2.4.7
Versión PHP	5.5.9
Versión Node.js	V0.10.25
Versión MySQL	5.5.43

Fuente: elaboración propia.

4.2. Requerimientos

Los requerimientos están relacionados con las subastas en línea que el sistema debe gestionar.

4.2.1. Gestión de los clientes

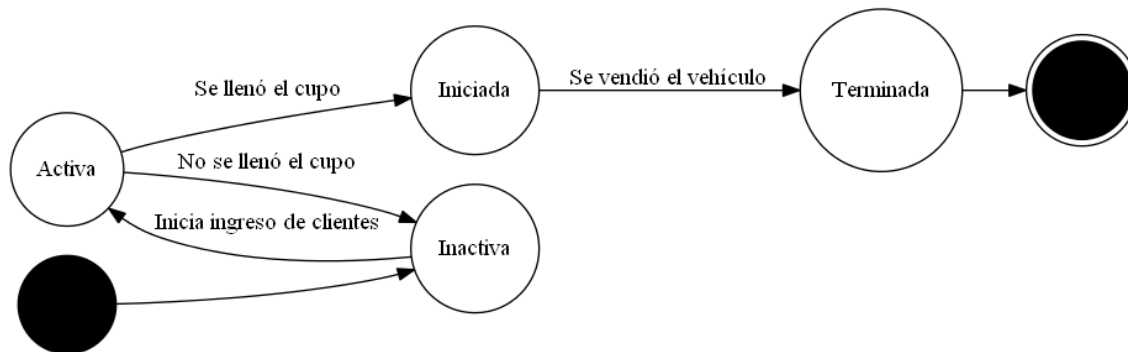
Todos los clientes registrados de la empresa pueden participar en las subastas en línea. La nueva aplicación debe poseer una pantalla de autenticación en la que el cliente ingrese su correo electrónico y contraseña para verificar que realmente es un cliente registrado.

4.2.2. Selección de subasta

Luego de que el usuario haya pasado el proceso de autenticación, se le debe mostrar el listado de las subastas disponibles para que seleccione la subasta a la que quiere ingresar. Para que un cliente pueda ingresar a una subasta, esta debe encontrarse en estado activo, en la figura 17 se muestra el diagrama de estados de una subasta. Una subasta puede estar en alguno de los siguientes estados:

- **Activa:** se encuentra en este estado cuando el administrador de la subasta, que es uno de los empleados de la empresa, indica al sistema que los usuarios pueden empezar a ingresar a dicha subasta.
- **Inactiva:** todas las subastas están por defecto en este estado, en el que la fecha y hora de la subasta ya fue fijada pero los usuarios no pueden ingresar a ella todavía. Es posible que ningún cliente desee participar en la subasta mientras está activa, si fuera este el caso, el administrador puede cambiar el estado de la subasta nuevamente a inactiva, para activarla en otro momento.
- **Iniciada:** una subasta pasa a este estado luego de estar activa, cuando el administrador de la subasta indica al sistema que dicha subasta debe iniciar. Todos los clientes participantes de la subasta, pueden realizar sus ofertas cuando la subasta se encuentra en este estado.
- **Terminada:** la subasta pasa a este estado, cuando se ha encontrado un cliente ganador que comprará el vehículo subastado.

Figura 17. **Diagrama de estados de una subasta**



Fuente: elaboración propia, empleando Graphviz.

4.2.3. **Gestión de subastas**

Uno de los empleados de la empresa cumplirá el papel de administrador de la subasta, él podrá iniciar una subasta o darla por terminada. Para ello, el sistema debe contar un una pantalla que permita gestionar los estados de las subastas. Además, el sistema debe validar que no se realice más de una subasta a la vez, porque el Departamento de Ventas ha observado que si se hacen varias subastas simultáneamente, se registran pérdidas porque la mayoría de los buenos clientes desearían participar en ambas subastas pero no pueden hacerlo.

4.2.4. **Participación en la subasta**

Este es el entorno más importante de la nueva aplicación, porque es donde cada cliente puede ver el progreso de la subasta y ofertar si así lo desea, además, se muestra el estado de la subasta y la información más general del vehículo subastado.

4.3. Solución propuesta

Para todo el sistema de subastas en línea se propone un diseño web responsivo que permitirá acceder a las páginas web desde cualquier dispositivo, sin alterar la calidad del diseño pues los elementos de las páginas web se adaptarán al tamaño del dispositivo. Se respetará el diseño actual de la base de datos existente y se agregarán los campos necesarios para realizar las subastas en línea. La solución se dividirá en cuatro módulos. Para cada uno de los módulos se muestran *mockups* de las páginas que se proponen, en ellas se muestran marcas, líneas y modelos de vehículos reales, así como datos ficticios sobre subastas que se realizarán, dicha información es utilizada en los diferentes *mockups* con fines ilustrativos.

4.3.1. Inicio de sesión

En este módulo se validará que los usuarios sean clientes registrados, este será desarrollado en PHP, al igual que la parte del sistema que ya existe, porque no necesita ninguna cualidad asíncrona, simplemente envía al servidor web los datos de la autenticación. Se hace la validación en la base de datos y el servidor responde al cliente el resultado de la validación, de ser correcta, se redirige al usuario hacia la página de selección de subastas, todo el diseño web será responsivo, por lo que las páginas se adaptarán al tamaño de la pantalla del dispositivo. Tanto el correo electrónico del cliente como su contraseña, son campos obligatorios en la base de datos y cada cliente tiene un correo único, el sistema no permite el registro de dos clientes con el mismo correo electrónico. El usuario podrá ingresar por primera vez al sistema con la opción de recuperar contraseña, posteriormente podrá modificar su contraseña. Las contraseñas serán encriptadas por el sistema con el algoritmo MD5 como medida de seguridad.

En el formulario de inicio de sesión, el usuario debe completar un captcha por seguridad, se utilizará reCAPTCHA en su segunda versión para proveer este servicio. Además, este formulario tendrá la opción de recuperar contraseña, en la figura 18 se muestra un *mockup* de la página de inicio de sesión visualizada en un teléfono inteligente.

Figura 18. **Mockup de la página de inicio de sesión visualizada en un teléfono inteligente**



Fuente: elaboración propia, empleando HTML5, CSS3 y reCAPTCHA.

4.3.2. Selección de subasta

La página de selección de subasta es la página a la que el usuario entra luego del inicio de sesión. Esta página muestra el conjunto de todas las subastas inactivas y la subasta activa en caso que esta existiera, esta pantalla no muestra las subastas iniciadas ni las terminadas. Si la subasta estuviera activa, entonces la página mostraría al usuario un botón con la opción de ingresar a la subasta, si la subasta estuviera inactiva simplemente se le mostraría al usuario la información de la subasta. Esta página debe mostrar en tiempo real las modificaciones que el administrador de subastas realice sobre el estado de las subastas.

Por ejemplo, cuando el administrador activa una subasta, la página debe mostrar al usuario el botón con la opción de participar automáticamente sin que se tenga que recargar la página. Para resolver esta situación se propone el uso de Websockets que reciban información de manera asíncrona por parte del servidor cada vez que se actualice el estado de una subasta, para que se modifique el contenido de la página de tal manera que se muestre el botón con la opción de participación en la subasta que se encuentre activa.

En la figura 19 se observa un *mockup* de la página de selección de subastas que muestra todas las subastas, al seleccionar alguna, se despliega la información de dicha subasta. En la pestaña de cada subasta se muestra la marca del vehículo, la línea, el año y el precio base de la subasta. Si se tratara de una subasta en estado activo se visualizaría un botón con la opción de participar, tal como lo muestra el *mockup* de la figura 20. Si se tratara de una subasta en estado inactivo, se visualizaría solamente la información de la subasta, tal como lo muestra el *mockup* de la figura 21.

Figura 19. **Mockup de la página de selección de subastas visualizada en un teléfono inteligente**



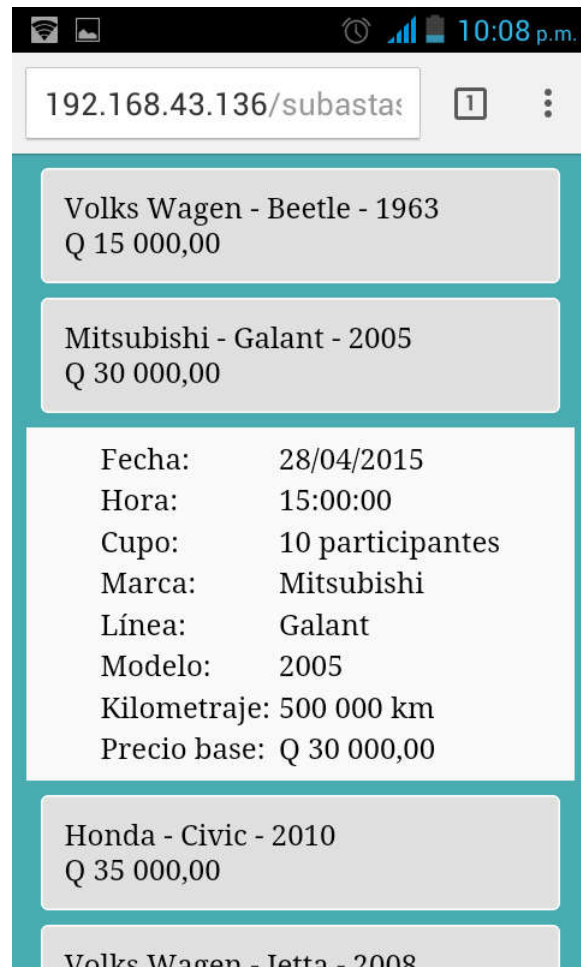
Fuente: elaboración propia, empleando HTML5 y CSS3.

Figura 20. **Mockup de la página de selección de subastas al seleccionar una subasta en estado activo, visualizada en un teléfono inteligente**



Fuente: elaboración propia, empleando HTML5 y CSS3.

Figura 21. **Mockup de la página de selección de subastas al seleccionar una subasta en estado inactivo, visualizada en un teléfono inteligente**



Fuente: elaboración propia, empleando HTML5 y CSS3.

4.3.3. Gestión de subastas

Solo los empleados tienen acceso a esta página, porque solo ellos pueden actualizar el estado de una subasta, a esta página se ingresa desde el sistema interno existente. Esta página se desarrollará en PHP y no cuenta con ninguna

característica de tiempo real, simplemente el usuario selecciona la opción que desee para modificar el estado de una subasta. Esta información se envía al servidor web que hace las modificaciones que debe en la base de datos y envía una respuesta al cliente. La página hace las validaciones correspondientes entre los estados y según el estado actual muestra en pantalla la opción que permite cambiar el estado. No es posible cambiar el estado a una subasta terminada porque este es el último estado por el que una subasta pasa, esto significa que la consistencia en el estado de las subastas se garantiza en esta aplicación.

En la figura 22 se muestra un *mockup* con el listado de todas las subastas disponibles. En la figura 23 se muestra un *mockup* con tres subastas desplegadas, cada una se encuentra en diferentes estado por lo que cada una posee una opción de cambio de estado diferente.

Figura 22. **Mockup de la página de gestión de subastas, visualizada en una tableta digital**



Fuente: elaboración propia, empleando HTML5 y CSS3.

Figura 23. **Mockup de la página de gestión de subastas con algunas subastas desplegadas, visualizada en una tableta digital**



Fuente: elaboración propia, empleando HTML5 y CSS3.

4.3.4. Monitor de subasta

En esta página los clientes podrán ver todo el progreso de la subasta y podrán realizar sus ofertas. Esta página es especialmente importante porque es donde la web en tiempo real se aplica para obtener los resultados deseados. Cada vez que un cliente realiza una oferta, todos los otros participantes deben conocer dicha oferta inmediatamente para que la dinámica de la subasta se desarrolle adecuadamente.

Para resolver lo anterior, se propone el uso de Node.js, ya que su rendimiento lo hace ideal para la tarea, y el módulo Socket.IO puede utilizarse para la comunicación asíncrona que debe existir entre los clientes y el servidor. Todos los clientes estarán conectados al servidor Node.js a través de *sockets* abiertos con el módulo Socket.IO. Cuando un cliente realice una oferta, esta información será enviada al servidor a través del *socket* y luego de recibirla, el servidor enviará la actualización a todos los *sockets* conectados, es decir, a todos los participantes de la subasta. En la figura 24 se muestra la apariencia que tendrá este módulo.

Figura 24. **Mockup de la página del monitor de subasta, visualizada en una tableta digital**



Fuente: elaboración propia, empleando HTML5 y CSS3.

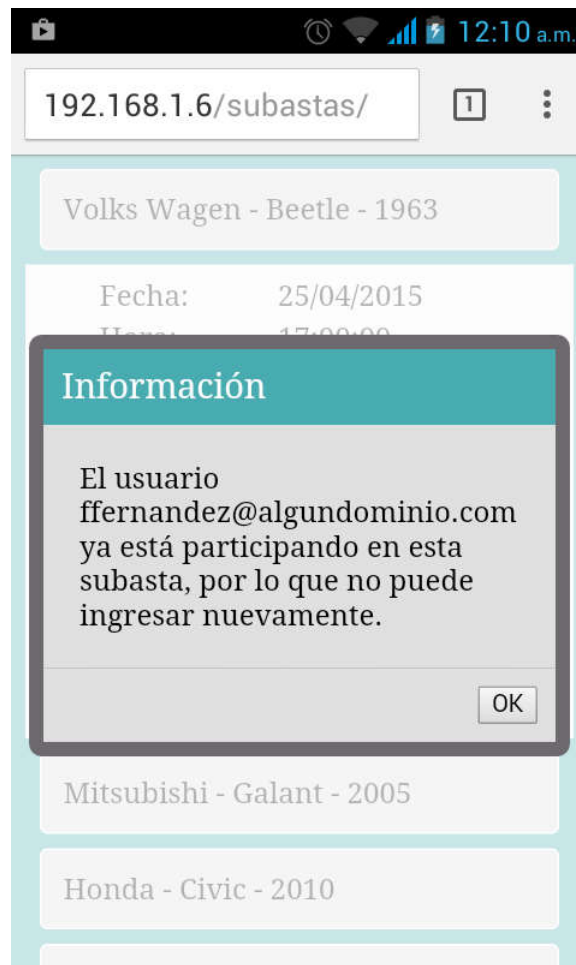
4.3.4.1. Solución a los posibles problemas de concurrencia

Un problema relacionado con la concurrencia que puede surgir, es aquel en el que un usuario realiza ofertas para una subasta desde dos sesiones diferentes. La única explicación para esto es que dos personas inicien sesión desde máquinas distintas, esto provoca que los otros participantes estén en desventaja y dentro de la lógica de la aplicación no debe estar permitido.

Para resolver este problema, se gestionará la participación única de los usuarios desde el monitor de subasta a través de los *sockets* utilizados, cada vez que un usuario ingresa al monitor de subasta se abre un *socket* con la librería Socket.IO para el intercambio de información con el servidor. Cada uno de estos *sockets* está asociado a un cliente específico, cuando el cliente intente abrir el *socket* el servidor verificará si ya existe un *socket* abierto para dicho cliente, de ser así, se desplegará un mensaje de advertencia como el que se muestra en la figura 25 y el usuario no saldrá de la página de selección de subasta.

Cada vez que el cliente cierre la ventana en la que se ejecuta el monitor de subasta, se cerrará el *socket* que utiliza, aunque no se haya cerrado sesión. Esto significa que el cliente podrá tener su sesión abierta en muchas computadoras, pero únicamente podrá abrir la ventana del monitor de subastas en una. Si el usuario dejara la ventana de subastas abierta en una sesión, entonces no podría utilizar este módulo en ninguna otra sesión, porque el sistema no tiene forma de validar que realmente no hay nadie en la ventana que se dejó abierta. Es por ello que en los manuales de usuario se hará énfasis en este aspecto, el usuario solamente puede tener un monitor de subastas abierto al mismo tiempo y solo desde este monitor puede ofertar.

Figura 25. **Mockup del mensaje de advertencia mostrado a los usuarios cuando intentan participar más de una vez en una subasta**



Fuente: elaboración propia, empleando HTML5 y CSS3.

De esta manera se provee el diseño una aplicación web en tiempo real que satisface las necesidades de la empresa de subastas, en la que el cliente establece comunicación asíncrona con el servidor mediante *sockets*, que se implementan con el módulo *Socket.IO* de *Node.js*, una de las tecnologías presentadas en capítulos anteriores, que resultó más eficiente que *Apache* en las pruebas de rendimiento realizadas en la sección 3.9. La aplicación muestra

a todos los usuarios la información de las múltiples ofertas realizadas en tiempo real, dando al usuario un servicio satisfactorio y eficiente.

CONCLUSIONES

1. Un sistema de comunicación asíncrona es mejor que un sistema de comunicación síncrona para el desarrollo de un sitio web en tiempo real, porque la comunicación entre el cliente y el servidor es más fluida, lo que permite un intercambio de información más eficiente y brinda un mejor rendimiento.
2. Ajax y Node.js son herramientas tecnológicas con las que se puede implementar eficientemente un sitio web en tiempo real, ya que permiten establecer comunicación asíncrona con el servidor. Esto proporciona importantes beneficios en el rendimiento, la accesibilidad, la interactividad y la experiencia del usuario en la aplicación web.
3. Las tecnologías web pueden ajustarse a las necesidades de un sistema de tiempo real cuando se utilizan las tecnologías adecuadas porque puede alcanzarse el rendimiento esperado.
4. Las múltiples ventajas de las aplicaciones web sobre las aplicaciones de escritorio las hacen una opción más práctica para el desarrollo de soluciones de software.
5. Al utilizar Ajax se obtienen aplicaciones web más interactivas en las que el usuario ya no debe esperar a que la página se recargue completamente para observar los resultados.

6. Node.js del lado del servidor permite realizar tareas en lapsos cortos, esto gracias a V8, el motor de JavaScript de alto rendimiento que utiliza, además, permite manejar altos niveles de concurrencia y establecer comunicación asíncrona con el cliente.

RECOMENDACIONES

1. Al desarrollar una aplicación web en tiempo real, deben tomarse en cuenta todos los factores que puedan repercutir en el rendimiento, especialmente el servidor web y el DBMS que se utilizará.
2. En las aplicaciones desarrolladas con Node.js, se debe evitar el código bloqueante porque se tiene un solo hilo de ejecución, si hay código bloqueante todo el sistema quedará suspendido hasta que se termine de ejecutar.
3. Para el desarrollo de aplicaciones con Ajax, se deben utilizar las librerías de JavaScript descritas en el presente trabajo, porque facilitan considerablemente el trabajo de codificación.
4. Si se desarrolla una aplicación se en tiempo real con Node.js, se recomienda el uso del módulo Socket.IO, para establecer la comunicación asíncrona entre el cliente y el servidor mediante *sockets*.

BIBLIOGRAFÍA

1. ABERNETHY, Michael. *¿Simplemente qué es Node.js?*. [en línea]. <<http://www.ibm.com/developerworks/ssa/opensource/library/os-nodejs/>>. [Consulta: marzo de 2015].
2. ÁLVAREZ, Miguel Ángel. *Introducción a XML*. [en línea]. <<http://www.desarrolloweb.com/manuales/18/>>. [Consulta: marzo de 2015].
3. BARRUETO, Luis Eduardo. *¿Qué es la web en tiempo real?: El caso Twitter*. [en línea]. <<http://www.maestrosdelweb.com/que-es-la-web-en-tiempo-real-el-caso-twitter/>>. [Consulta: marzo de 2015].
4. EGUILUZ, Javier. *Introducción a Ajax*. [en línea]. <<http://librosweb.es/libro/ajax/>>. [Consulta: marzo de 2015].
5. GARRET, Jesse James. *Ajax: A new approach to web applications*. [en línea]. <<http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications/>>. [Consulta: marzo de 2015].
6. HOLDENER, Anthony T. *Ajax: The Definitive Guide*. California: O'Reilly, 2008. 957 p. ISBN 978-0-596-52838-6.

7. MAESTROS DEL WEB. *Ajax: Un nuevo acercamiento a las aplicaciones web.* [en línea]. <<http://www.maestrosdelweb.com/ajax/>>. [Consulta: marzo de 2015].
8. MOCIONSOFT. *¿Por qué usamos Node.js?.* [en línea]. <<http://mocionsoft.com/blog/porque-usamos-node-js/>>. [Consulta: marzo de 2015].
9. NOMBELA, Nicolas. *Asincronicidad en Node.js.* [en línea]. <<http://nnombela.com/blog/2012/03/21/asincronicidad-en-node-dot-js/>>. [Consulta: marzo de 2015].
10. NOVAS, Pablo. *WebSockets y Socket.IO.* [en línea]. <<http://fernetjs.com/2012/11/websockets-y-socketio/>>. [Consulta: marzo de 2015].
11. PASTOR, Francisco. *Apuntes de la asignatura Sistemas informáticos de tiempo real.* [en línea]. <http://www.uv.es/gomis/Apuntes_SITR/>. [Consulta: marzo de 2015].
12. PLAZA, Fernando. *Comunicación síncrona vs comunicación asíncrona.* [en línea]. <<http://www.fernandoplaza.com/2009/03/comunicacion-sincrona-vs-comunicacion-asincrona.asp>>. [Consulta: marzo de 2015].
13. RAVULAVARU, Arvind. *What is Node.js.* [en línea]. <<http://thejackalofjavascript.com/nodejs/>>. [Consulta: marzo de 2015].

APÉNDICES

Apéndice 1. **Ejemplo de uso de Node.js en el desarrollo de una aplicación web en tiempo real**

Descripción de la aplicación

La aplicación a desarrollar es un monitor de procesos que desplegará la información de los múltiples procesos que se están ejecutando en cierta computadora; será desarrollado en un entorno web y se podrá acceder a él desde cualquier dispositivo que se conecte a la red en la que se encuentra conectada la computadora. Para lograrlo, se utilizará Node.js como servidor, dicho servidor estará consultando los procesos de la computadora con cierta frecuencia y le enviará estos datos a la aplicación cliente mediante *sockets* con ayuda del módulo Socket.IO de Node.js. Será una aplicación en tiempo real porque el usuario tendrá acceso a la información de los procesos inmediatamente después de que el servidor hace la consulta, además, los clientes tendrán una opción que les permitirá finalizar el proceso que deseen. Las especificaciones de software y hardware de la computadora utilizada para el desarrollo de la aplicación se muestran en la tabla I.

Tabla I. **Especificaciones de software y hardware de la computadora utilizada en el desarrollo de la aplicación**

Característica	Soporte
Memoria RAM	2 GB
Disco duro	500 GB
Procesador	Celeron (R) Dual-Core T3300 (2.00 GHz)
Sistema operativo	Ubuntu 14.04 LTS
Versión Node.js	V0.10.25

Fuente: elaboración propia.

Aplicación Node.js

El código correspondiente al archivo server.js se muestra en las próximas tres figuras. La figura 1 muestra el primer segmento de código Node.js, en él se declaran las variables, se les asigna un valor inicial haciendo referencia a módulos específicos y se escriben los métodos que envían los archivos relacionados con la página al cliente, además, se arranca el servidor para que escuche a través del puerto siete mil.

Figura 1. **Primer segmento de código aplicación en Node.js**

```
1  var app = require('express')();
2  var http = require('http').Server(app);
3  var io = require('socket.io')(http);
4  var spawn = require('child_process').spawn;
5
6  //función que envía al cliente el archivo index.html
7  app.get('/', function(req, res){
8    res.sendFile(__dirname + '/index.html');
9  });
10
11 //función que envía al cliente el archivo style.css
12 app.get('/style.css', function(req, res){
13   res.sendFile(__dirname + '/style.css');
14 });
15
16 //el servidor empieza a escuchar a través del puerto 7000
17 http.listen(7000, function(){
18   console.log('Servidor escuchando en el puerto: 7000.');
```

Fuente: elaboración propia, empleando Notepad Plus Plus v6.7.7.

En la figura 2 se muestra el segundo fragmento de código de la aplicación Node.js. En él se desarrolla la función que se utiliza para la conexión de los clientes, así como la que se utiliza para enviar información de actualización de procesos cada seis segundos y la función que sirve para finalizar los procesos que el cliente escoja.

Figura 2. Segundo segmento de código aplicación en Node.js

```
20
21 var procedimientosLeidos;
22 //función que se dispara cuando un nuevo cliente se conecta
23 io.sockets.on('connection', function(socket){
24     console.log('Un usuario nuevo ha establecido conexión.');
```

25 //variable que recoge los datos de los procesos: top -b -d 6
26 //con -b se recoge el texto y con -d se le da un retardo de 6 segundos
27 var cmd = spawn('top', ['-b', '-d', '6']);
28 cmd.stdout.on('data', function(data){
29 var txt = new Buffer(data).toString('utf8', 0, data.length);
30 txt=limpiarTexto(txt.trim());
31 var indiceOrden = txt.lastIndexOf('ORDEN');
32 var tamañoCadena = data.length;
33 if((tamañoCadena < 4096) && (tamañoCadena != 142)){
34 socket.emit('monitorProcesos', { procesos: procedimientosLeidos});
35 }else{
36 if(indiceOrden > 0){
37 procedimientosLeidos=txt.substring(indiceOrden+6,tamañoCadena);
38 }else{
39 procedimientosLeidos+=txt;
40 }
41 }
42 });
43
44 //función que se dispara cuando se manda a finalizar un proceso
45 socket.on('finalizarProceso', function(data){
46 console.log('finalizar Proceso');
47 spawn('kill', [data]);
48 });
49 });

Fuente: elaboración propia, empleando Notepad Plus Plus v6.7.7.

Por último, se desarrolla una función con el objetivo de limpiar las cadenas que contienen la información de los múltiples procesos, no solo para reducir la cantidad de información enviada, sino para facilitar al cliente la tarea de analizar el texto, dicha función se observa en la figura 3.

Figura 3. Tercer segmento de código aplicación en Node.js

```
50
51 //función que transforma varios espacios sucesivos en un solo espacio en blanco
52 function limpiarTexto(txt){
53     var txtNuevo = '';
54     var bandera=false;
55     for (var i = 0; i < txt.length; i++) {
56         if (!bandera){
57             txtNuevo=txtNuevo+txt[i];
58         }else{
59             if(txt[i]!=' '){
60                 txtNuevo=txtNuevo+txt[i];
61             }
62         }
63         if(txt[i]==' '){
64             bandera=true;
65         }else{
66             bandera=false;
67         }
68     }
69     return txtNuevo;
70 }
```

Fuente: elaboración propia, empleando Notepad Plus Plus v6.7.7.

Aplicación cliente

Se programó un cliente que se conecta al servidor mediante `sockets` con la ayuda de la librería `Socket.IO`. Para presentar el código de la página `index.html` se dividió el código en tres fragmentos, el primero de ellos se muestra en la figura 4, en él se puede ver el encabezado de la página así como la declaración y conexión del `socket` que se usará para el intercambio de información. También se muestra la función que se dispara cuando se recibe información de los procesos, dentro de la cual se actualiza la tabla que contiene los procesos.

Figura 4. Primer segmento de código aplicación cliente

```
1 <!DOCTYPE html>
2 <html lang="es">
3 <head>
4 <meta charset="utf-8"/>
5 <title>Monitor</title>
6 <link rel="stylesheet" href="style.css"/>
7 </head>
8 <body>
9 <center>
10 <h1>Monitor de Procesos</h1>
11 <p id="listaProcesos"></p>
12 </center>
13 <script src="socket.io/socket.io.js"></script>
14 <script>
15 //variable usada para conectarse al servidor en el puerto 7000
16 var socket = new io.connect('192.168.1.6', {port: 7000});
17 //función que se dispara cuando se reciben datos sobre los procesos
18 socket.on('monitorProcesos', function(data){
19     setProcesos(data);
20 });
```

Fuente: elaboración propia, empleando Notepad Plus Plus v6.7.7.

En la figura 5 se muestra el segundo fragmento de la página cliente, en él está el método que actualiza la tabla con el contenido de los procesos, esta actualización se hace mediante JavaScript y DOM, por lo que no es necesario recargar toda la página para actualizar este contenido.

Figura 5. Segundo segmento de código aplicación cliente

```
21 //función que es llamada cuando se necesita actualizar en el monitor
22 //la información recibida de los procesos
23 function setProcesos(procesos) {
24     var txt=procesos.procesos.trim();
25     var txtHtml=' <table> <tr> '+
26     ' <th>PID</th> <th>USUARIO</th> <th>PR</th> <th>NI</th> '+
27     ' <th>VIRT</th> <th>RES</th> <th>SHR</th> <th>S</th> '+
28     ' <th>%CPU</th> <th>%MEM</th> <th>HORA</th> <th>ORDEN</th>'+
29     ' <th>KILL</th> </tr> ';
30     var procs=txt.split('\n');
31     for(var i=0;i<procs.length;i++){
32         var elm=procs[i].trim().split(' ');
33         txtHtml+=' <tr> ';
34         if(elm.length==12){
35             for(var j=0;j<elm.length;j++){
36                 txtHtml+=' <td id="">'+elm[j].trim()+</td> ';
37             }
38             txtHtml+=' <td> ';
39             txtHtml+=' <input type="button" class="btn" onclick='+
40             '"finalizar('+elm[0]+');" value="Finalizar Proceso"/>';
41             txtHtml+=' </td> ';
42         }
43         txtHtml+=' </tr> ';
44     }
45     txtHtml+=' </table> ';
46     document.getElementById('listaProcesos').innerHTML=txtHtml;
47 }
```

Fuente: elaboración propia, empleando Notepad Plus Plus v6.7.7.

Por último, en la figura 6 se muestra el segmento final del código de la página cliente, en él está la función que se ejecuta cuando el cliente desea finalizar alguno de los procesos mostrados en el monitor. Para ello, se emite un mensaje al servidor que lo recibe y ejecuta las acciones correspondientes, el usuario solamente podrá finalizar los procesos para los cuales tiene permisos y los permisos que tenga dependen de la manera en que se haya ejecutado el servidor Node.js. Si este es ejecutado como superusuario, entonces será posible para el cliente finalizar cualquier proceso, por otro lado, si no se ejecuta con permisos de superusuario los procesos que podrá finalizar estarán restringidos a los que podría finalizar el usuario normal.

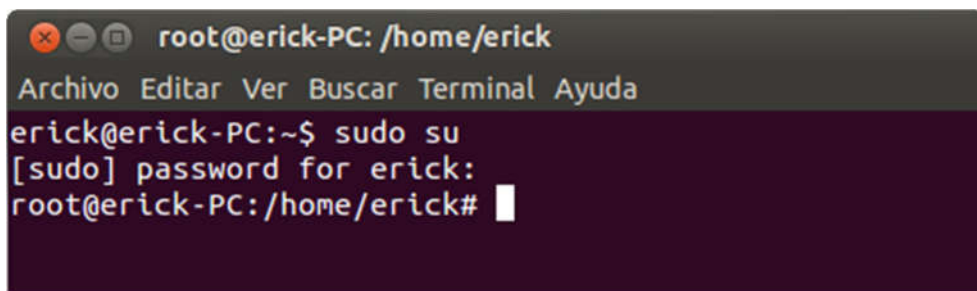
Figura 6. Tercer segmento de código aplicación cliente

```
48
49 //función que es llamada cuando el cliente quiere finalizar un proceso
50 function finalizar(vari){
51     socket.emit('finalizarProceso', vari);
52     alert("El proceso: "+vari+" se ha finalizado.");
53 }
54 </script>
55 </body>
56 </html>
```

Fuente: elaboración propia, empleando Notepad Plus Plus v6.7.7.

Un superusuario en Ubuntu y el resto de las distribuciones de GNU/Linux, es el usuario que posee todos los permisos de lectura, escritura y ejecución en el sistema operativo, habitualmente se le da el nombre de *root* y está por encima del usuario administrador, de hecho, pueden tenerse muchos usuarios administradores, pero solo un superusuario. Para activar la cuenta de superusuario en Ubuntu 14.04 LTS, se utiliza el comando `sudo su`, seguido de la contraseña del usuario, tal como se muestra en la figura 7.

Figura 7. Comando para activar la cuenta de superusuario



```
root@erick-PC: /home/erick
Archivo Editar Ver Buscar Terminal Ayuda
erick@erick-PC:~$ sudo su
[sudo] password for erick:
root@erick-PC:/home/erick#
```

Fuente: elaboración propia, empleando Ubuntu 14.04 LTS.

En la figura 8 se muestra el código correspondiente archivo style.css que da el estilo de la página cliente, este se agrega para obtener una página con un diseño más agradable.

Figura 8. **Código correspondiente al estilo de aplicación cliente**

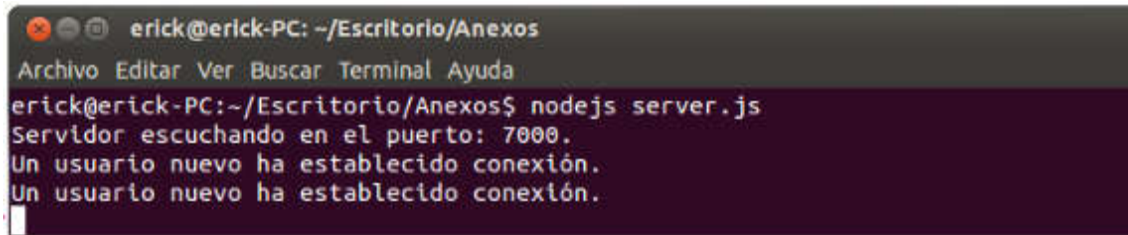
```
1  table {  
2      background: #f5f5f5;  
3      font-size: 13px;  
4      border-radius: 4px;  
5  }  
6  th {  
7      color: white;  
8      background: #6E696E;  
9  }  
10 td {  
11     border-right: 1px solid #fff;  
12     border-left: 1px solid #fff;  
13     border-top: 1px solid #fff;  
14     border-bottom: 1px solid #fff;  
15     padding: 0px 10px;  
16 }
```

Fuente: elaboración propia, empleando Notepad Plus Plus v6.7.7.

Ejecución de la aplicación Node.js

Para ejecutar la aplicación Node.js en Ubuntu 14.04 LTS, se ejecuta el comando `nodejs`, que recibe como parámetro la ruta absoluta del archivo que contiene el código fuente o simplemente el nombre del archivo si se encontrara en el mismo directorio que el usuario que ejecuta. En la figura 9 se muestra la ejecución de la aplicación Node.js, cuando el servidor arranca se muestra el mensaje de que se está escuchando en el puerto siete mil. Cada vez que un nuevo cliente se conecta, se muestra un mensaje de que se ha establecido una nueva conexión.

Figura 9. **Ejecución del archivo server.js**



```
erick@erick-PC: ~/Escritorio/Anexos
Archivo Editar Ver Buscar Terminal Ayuda
erick@erick-PC:~/Escritorio/Anexos$ nodejs server.js
Servidor escuchando en el puerto: 7000.
Un usuario nuevo ha establecido conexión.
Un usuario nuevo ha establecido conexión.
```

Fuente: elaboración propia, empleando Ubuntu 14.04 LTS.

Ejecución de la aplicación cliente

La aplicación puede ejecutarse desde cualquier navegador que tenga soporte para Socket.IO, la aplicación cliente fue utilizada exitosamente en el navegador Google Chrome versión 42.0.2311.135 m. En la figura 10 se muestra una imagen del cliente ejecutándose.

Figura 10. Ejecución del cliente en el navegador

PID	USUARIO	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	HORA	ORDEN	KILL
2990	erick	20	0	294460	42960	25232	S	3.8	2.2	0:01.72	chrome	Finalizar Proceso
1133	root	20	0	114680	22652	16316	S	3.3	1.1	0:08.37	Xorg	Finalizar Proceso
1954	erick	20	0	167244	22772	15672	S	2.8	1.1	0:04.88	gnome-panel	Finalizar Proceso
2382	erick	20	0	77928	11556	4996	S	1.3	0.6	0:02.24	zeitgeist-+	Finalizar Proceso
2790	erick	20	0	644300	80032	49800	S	1.3	4.0	0:03.46	chrome	Finalizar Proceso
1831	erick	20	0	165672	24216	11248	S	0.8	1.2	0:00.59	unity-sett+	Finalizar Proceso
2425	root	20	0	3528	1444	760	S	0.8	0.1	0:02.19	mount.ntfs	Finalizar Proceso
1803	erick	20	0	47508	3756	3000	S	0.7	0.2	0:02.88	ibus-daemon	Finalizar Proceso
2829	erick	20	0	337140	56596	20448	S	0.7	2.8	0:00.85	chrome	Finalizar Proceso
1931	erick	20	0	125056	15060	10496	S	0.5	0.8	0:01.61	metacity	Finalizar Proceso
1976	erick	20	0	239712	36352	22368	S	0.5	1.8	0:09.94	nautilus	Finalizar Proceso
13	root	20	0	0	0	0	S	0.3	0.0	0:00.10	ksoftirqd/1	Finalizar Proceso
1782	erick	20	0	4876	1920	940	S	0.3	0.1	0:00.61	dbus-daemon	Finalizar Proceso
1929	erick	20	0	128280	17556	7916	S	0.3	0.9	0:00.77	bamfdamon	Finalizar Proceso
3004	erick	20	0	6740	1468	1096	R	0.3	0.1	0:00.08	top	Finalizar Proceso
3	root	20	0	0	0	0	S	0.2	0.0	0:00.12	ksoftirqd/0	Finalizar Proceso
7	root	20	0	0	0	0	S	0.2	0.0	0:00.32	rcu_sched	Finalizar Proceso
72	root	20	0	0	0	0	S	0.2	0.0	0:01.95	kworker/1:2	Finalizar Proceso
1037	mysql	20	0	319392	41644	5756	S	0.2	2.1	0:00.54	mysqld	Finalizar Proceso
1841	erick	20	0	29152	4940	2508	S	0.2	0.2	0:00.05	gvfsd	Finalizar Proceso
1860	erick	20	0	122744	17428	10068	S	0.2	0.9	0:00.92	ibus-ui-gt+	Finalizar Proceso

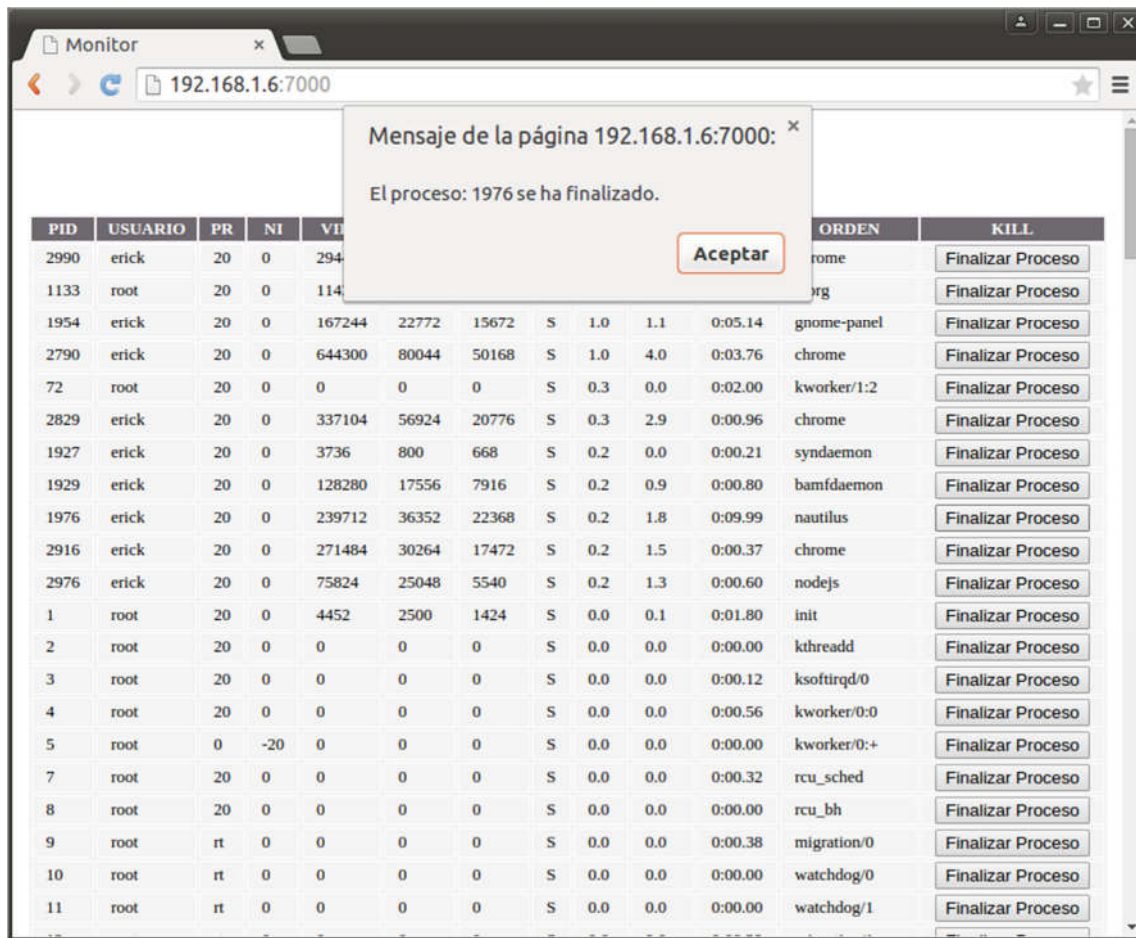
Fuente: elaboración propia, empleando Google Chrome versión 42.0.2311.135 m. y Ubuntu 14.04 LTS.

Finalizar procesos desde el cliente

La aplicación cliente tiene la opción de finalizar los procesos, según los permisos que se le hayan dado al *server* cuando fue ejecutado, por ejemplo, en la ejecución del servidor Node.js que se mostró anteriormente, no se dieron permisos de superusuario, por lo que el cliente no podrá finalizar los procesos

que requieran permisos de este tipo. Luego de finalizar un proceso exitosamente, se despliega un mensaje como el mostrado en la figura 11.

Figura 11. Mensaje mostrado después de finalizar un proceso



Fuente: elaboración propia, empleando Google Chrome versión 42.0.2311.135 m. y Ubuntu 14.04 LTS.