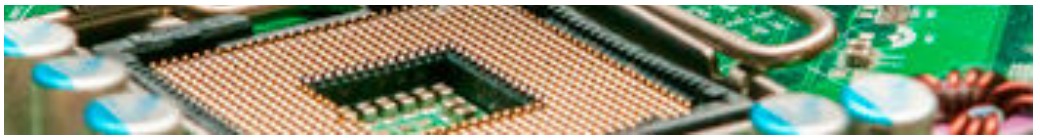
Servicios y Aplicaciones Distribuidas

2014

Seminario 4 | Express y Websockets

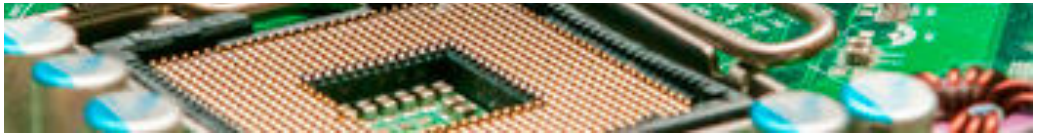
MUIInf

SAD

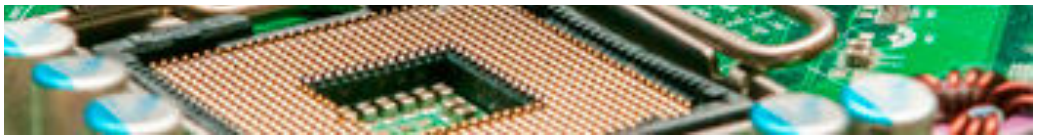


- Express
  - Preliminares
  - El módulo http
  - El módulo connect
  - El módulo express
- Websockets

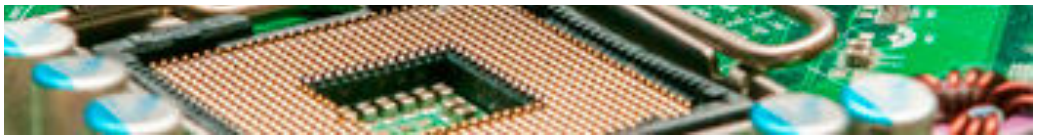
- Express.js, o simplemente **express**, es un módulo de Node.js que introduce un framework para el desarrollo de aplicaciones web, en la parte del servidor, incluyendo el patrón de diseño MVC (Modelo-Vista-Controlador).
- El desarrollo de aplicaciones web de baja complejidad se puede realizar directamente a partir del módulo http de Node.js.
- Sin embargo, tanto si esta complejidad es alta como baja el uso de express facilita enormemente este desarrollo.
- Express se basa en el módulo connect y éste en el módulo http.



- El módulo **http** proporciona servicios básicos para el desarrollo de servidores web, centrándose en el manejo de las peticiones y respuestas.
- **Connect** es para express un módulo middleware, que a su vez utiliza al módulo http como middleware. Connect facilita:
  - El uso de cookies y de sesiones,
  - El procesamiento del contenido de los métodos-HTTP,
  - La compresión de datos,
  - Autenticación, etc.
- **express** introduce el manejo de rutas y el uso de plantillas (en ejs, jade, etc.) para mostrar los resultados en el navegador.

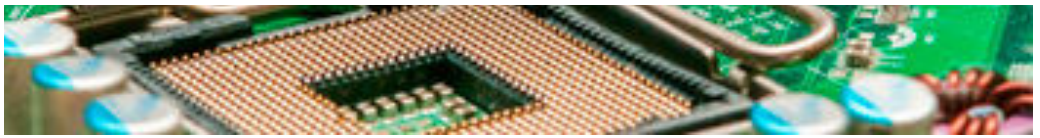


- El Hypertext Transfer Protocol o simplemente http es el protocolo utilizado en las transacciones en la web.
- Es un protocolo sin estado orientado a transacciones en el contexto cliente-servidor. El cliente realiza la solicitud de un servicio al servidor y espera su respuesta.
- Este servicio puede ser:
    - descargar o cargar un archivo,
    - realizar una computación vía el servidor,
    - consultar una base de datos, etc.

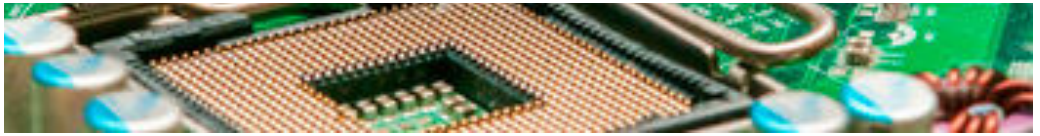


El protocolo HTTP:

- La comunicación entre el cliente y el servidor se lleva a cabo mediante mensajes estructurados.
- Cada mensaje se compone de un encabezamiento y opcionalmente de un cuerpo.
- En el encabezamiento básicamente aparece la información de control y la asociada a la URL.
- En el cuerpo aparecen los datos propiamente dichos del mensaje.
- Los servicios a los que se dirigen la solicitudes estarán referenciados mediante URLs.

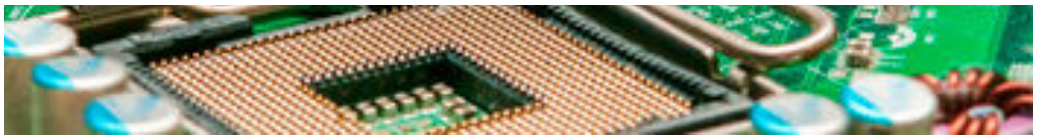


- La comunicación siempre la inicia el cliente mediante un mensaje que corresponde con un método-HTTP (también llamados verbos).
  - GET: se utiliza para la descargas de los recursos identificados por la URL. Por ejemplo descargar archivos. Puede también enviar una pequeña cantidad de información al servidor en la query de la URL.
  - POST: se utiliza para enviar datos al servidor en el cuerpo del mensaje. Su URL identifica al recurso que tiene que procesar el servidor. Por ejemplo los datos de un formulario.
  - PUT: permite que en el servidor se guarde la información enviada. Su URL identifica al contenido. Se utiliza para subir archivos al servidor.
  - DELETE: se utiliza para eliminar recursos en el servidor.



Un servidor realizado mediante el módulo http de node procesará la solicitud del cliente y le enviará una respuesta.

- En este proceso, se dispone de dos objetos:
  - Uno asociado a la petición, habitualmente denominado request o req: contiene toda la información del mensaje del cliente de modo que el servidor pueda fácilmente procesarla.
  - Otro asociado a la respuesta, habitualmente denominado response o res: permite al servidor construir sobre el mismo todo el mensaje respuesta para luego ser enviado.

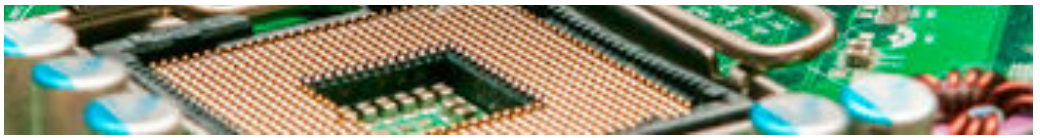


Express - Módulo http. Ejemplo

```
//Hola mundo
var http = require("http");

function onRequest(request, response) {
        console.log("Peticion Recibida.");
        response.writeHead(200, {"Content-Type": "text/html"});
        response.write("Hola Mundo");
        response.end("<br>Fin");
}

http.createServer(onRequest).listen(10000,function(){
        console.log("Servidor Iniciado.");
})
```

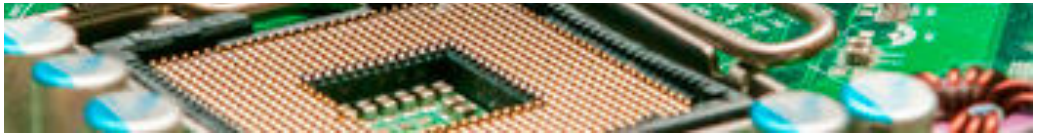



Express - Módulo connect

- El módulo connect es un middleware que permite, mediante la secuenciación de sus módulos, la creación de una pipeline a lo largo de la cual se procesan los objetos request y response de la solicitud.

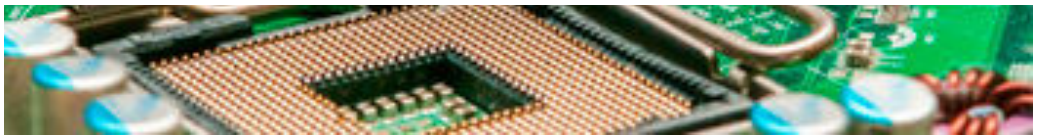
- Utiliza al módulo http como middleware.

- Este procesamiento se ve enormemente facilitado por el soporte proporcionado por los módulos que connect pone a nuestra disposición.

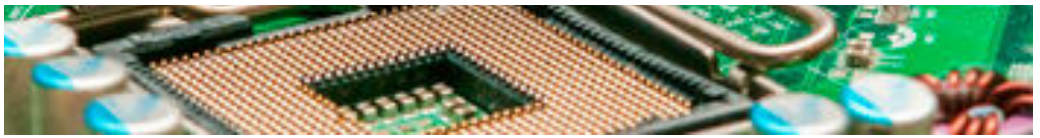


Entre estos módulos se encuentran los siguientes (véase [2]):

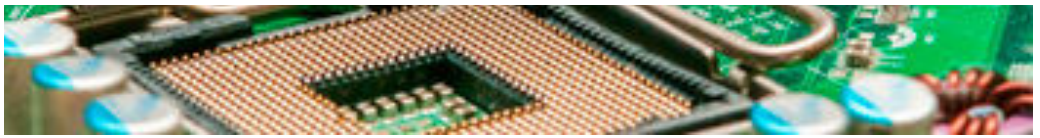
- **logger()**: permite registrar de acuerdo a diferentes formatos la información relativa a la solicitud que se procesa. El registro puede hacerse en el stdout o en un stream dado.
- **bodyParser()**: se encarga de analizar los datos del mensaje para obtener request.body y request.file para su posterior uso en la pipeline.
- **json()**: realiza el parsing de application/json
- **urlencoded()**: realiza el parsing de application/x-www-form-urlencoded.
- **query():** proporciona request.query, que es la query de la url, para su posterior uso en la pipeline.



- **favicon()**: responde a la solicitud /favicon.ico.
- **cookieParser()**: se encarga de procesar las cookies. Admite un string que actúa como una palabra secreta para la firma de las cookies para su posterior uso en la pipeline. Se utiliza para firmar la cookie de la sesión.
- **session()**: permite definir sesiones persistentes por medio de request.session entre el cliente y el servidor a través del uso de cookies firmadas. Depende de cookieParser.
- **cookieSession()**: permite definir sesiones persistentes donde además los elementos de la sesión se envían como cookies firmadas al cliente en formato json.



- **methodOverride()**: permite procesar solicitudes que no se corresponden con el método-HTTP utilizado. Esto se debe a que los navegadores sólo utilizan los métodos-HTTP GET y POST. Así, por ejemplo, puede procesarse un método POST como si fuera un PUT.
- **vhost()**: permite sobre un mismo servidor definir múltiples hosts virtuales.
- **static()**: directorio de los ficheros públicos que el servidor puede servir de modo automático.
- Otros módulos de interés: **errorHandler()**, **directory()**, **staticCache()**, **responseTime()**, **timeout()**, **basicAuth()**, **compress()**, etc.

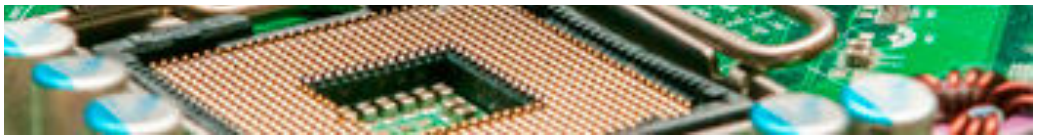


Express - Módulo connect. Ejemplo.

- //Se utiliza la funcion **next()** para pasar el control al siguiente modulo de la pipeline

```
var connect = require('connect');
var n = 0;
function registro(req, res, next) {
        console.log(req.method + "   " + req.url);
        next();
};
function hola(req, res) {
        res.setHeader('Content-Type', 'text/plain');
        res.end('hola mundo  ' + n);
};

function acceso(req, res, next){
        console.log("acceso  " + n++);
        next();
};
```

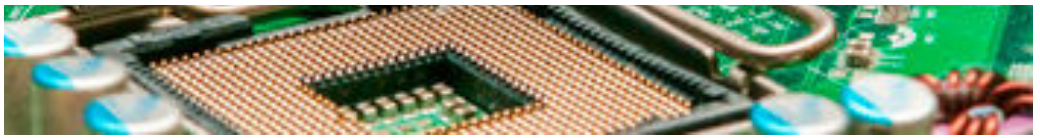
Express - Módulo connect. Ejemplo.

```
//pipeline

connect()
.use(registro)
.use(acceso)
.use(hola)

.listen(10000, function(){
        console.log("Servidor en servicio");
});
```

- La función next() permite que continúe la ejecución de la pipeline en el siguiente módulo de la secuencia. Si no aparece, la ejecución termina en ese módulo.
- En cualquier caso, en el momento en que se envía la respuesta al cliente ya no se continúa con la ejecución de la pipeline.



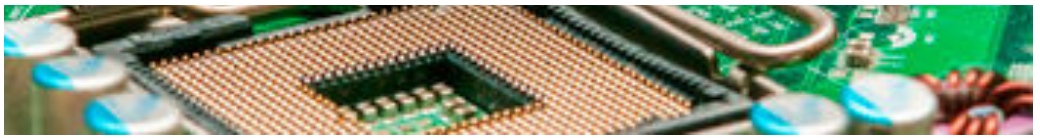
- El módulo express tiene como middleware al módulo connect de modo que todo lo que anteriormente hemos referenciado como

  connect.modulo()

- ahora podemos hacerlo como

  express.modulo().

- Express, por tanto, tiene todo lo que tiene connect y además introduce a su vez:
  - Funciones propias.
  - Control de rutas.
  - Plantillas para facilitar la descripción de la representación de la información en el navegador (ejs, jade, etc.).



Express - Módulo express

- Express, puede instalarse como módulo, mediante npm, de forma local o de forma global.

- La instalación local se realiza

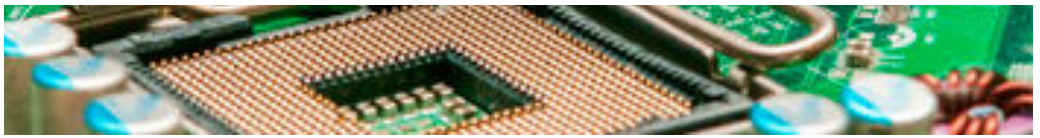
  npm install express

- La instalación global, mediante '-g'

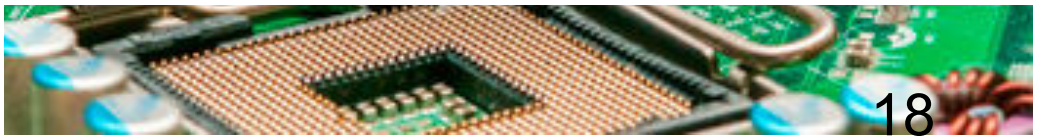
  npm install –g express

- Con la instalación global, también dispondremos del la orden 'express', que podrá utilizarse para crear una estructura de directorios genérica para hacer aplicaciones con express.

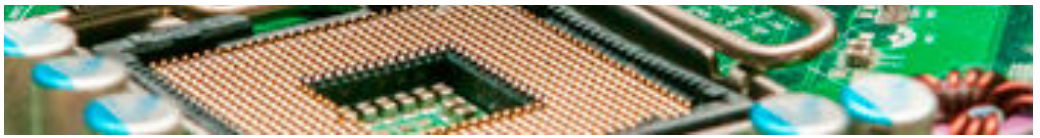
  express aplicación



- Express
  - Preliminares
  - El módulo http
  - El módulo connect
  - El módulo express
- **Websockets**

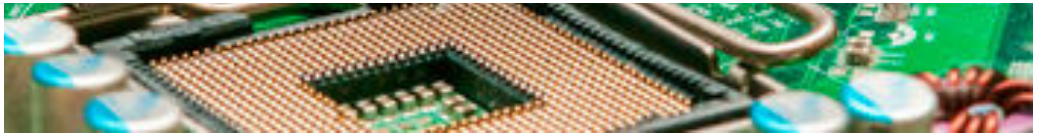


- Implementación de múltiples canales bidireccionales, full-duplex sobre una conexión TCP sobre el puerto 80.
- Permite acceder a servidores usando únicamente el puerto 80 que resulta ser en la práctica el único puerto que mantienen abiertos muchos ordenadores de Internet.
- Está diseñado para conexiones entre navegadores y servidores web, pero se puede emplear para cualquier aplicación cliente/servidor
- Se trata de un estándar W3C, IETF
- Se basa en el protocolo HTTP y la extensión HTTP upgrade.
- Introduce una pequeña sobrecarga a una mera conexión TCP, por la negociación inicial hasta el establecimiento de la conexión.



Ejemplo de negociación:

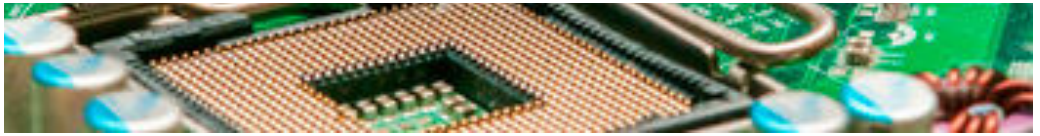
Petición cliente	Respuesta servidor
GET /demo HTTP/1.1 Host: example.com Connection: Upgrade Sec-WebSocket-Key2: 12998 5 Y3 1 .P00 Sec-WebSocket-Protocol: sample Upgrade: WebSocket Sec-WebSocket-Key1: 4 @1 46546xW%0l 1 5 Origin: http://example.com ^n:ds[4U	HTTP/1.1 101 WebSocket Protocol Handshake Upgrade: WebSocket Connection: Upgrade Sec-WebSocket-Origin: http://example.com Sec-WebSocket-Location: ws://example.com/ demo Sec-WebSocket-Protocol: sample 8jKS'y:G*Co,Wxa-



- Se utiliza el prefijo de protocolo ws:// y wss:// para las conexiones sin cifrar y cifradas respectivamente

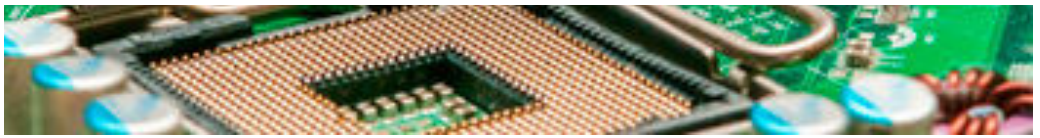
- Está integrado su uso en javascript, y los navegadores y servidores recientes lo soportan.

- Cuando las conexiones atraviesan proxies, cortafuegos, etc, las conexiones sin cifrar fallan con cierta frecuencia a día de hoy por incompatibilidad de los nodos intermedios.



Websockets - ejemplo

```
<meta charset="utf-8" />
<title>Test</title>
<script>
function testWS() {
          websocket = new WebSocket("ws://www.ejemplourl.org/");
          websocket.onopen = function(evt) {
                    websocket.send("Hola mundo");
          };
          websocket.onclose = function(evt) {
                    console.log ("desconectado");
          };
          websocket.onmessage = function(evt) {
                    console.log ("mensaje de entrada: " + evt.data);
                    websocket.close();
          };
          websocket.onerror = function(evt) {
                    console.log ("Error: " + evt.data);
          };
}
window.addEventListener("load", testWS, false);
</script>
<h2>Test simple WebSocket</h2>
```

- Express:
  - Sitio oficial: www.expressjs.com

- Websockets:
  - http://www.w3.org/TR/websockets/
  - Otros: wikipedia, www.websockets.org