Introducción a HTTP

Desarrollo de Aplicaciones Web Avanzado

Favio Náquira Vargas

- Es importante conocer la anatomía de una transacción HTTP. Así podremos utilizar headers, cookies y demás métodos para controlar toda petición que llegue a nuestro servidor.

Creación de un servidor

```
var http = require('http');

var servidor = http.createServer(function(request, response) {
  // la magia sucede aqui!
});
```

Creación de un servidor

- La función **createServer** es llamada cada vez que una solicitud HTTP es realizada al servidor, por lo que es llamada **request handler** (manejador de solicitudes).
- De hecho, el objeto servidor retornado por **createServer** es un **EventEmitter** (emisor de eventos) por lo que se le pueden agregar **listener** (escucha) posteriormente.

Creación de un servidor

```
var servidor = http.createServer();
servidor.on('request', function(request, response) {
  // el mismo tipo de magia sucede aqui!
});
```

Method, URL y headers

- Cuando se maneja una solicitud, lo primero que se suele hacer es mirar al método y a la URL, para poder decidir las acciones a realizar. Node facilita estas consultas agregando propiedades al objeto **request**.

```
var method = request.method;
var url = request.url;
```

Method, URL y headers

- Lo mismo aplica para **headers**. Node facilita su tratamiento gracias a que representa todos los **headers** en minúsculas, evitándonos el tortuoso unificamiento.

```
var headers = request.headers;
var userAgent = headers['user-agent'];
```

Request Body

- Cuando recibimos una petición POST o PUT, el request body (cuerpo de la solicitud) será lo más importante para nuestra aplicación. El objeto request implementa la interfaz ReadableStream, que puede ser escuchada o diseccionada como cualquier otro flujo de datos. Cada chunk (pedazo) emitido en el evento data es un Buffer

```
var body = [];
request.on('data', function(chunk) {
  body.push(chunk);
}).on('end', function() {
  body = Buffer.concat(body).toString();
  // ahora, body tiene todo el cuerpo de la peticion concatenado
});
```

Errores

- Como el objeto request es un ReadableStream, también es un EventEmitter y se comporta como uno cuando un error sucede.
- Si uno no tiene un listener para los errores, el error será arrojado durante la ejecución del programa (thrown) lo que originará que Node colapse (crash of Node)

```
request.on('error', function(err) {
  // Esto imprime la pila de errores sucedidos
  console.error(err.stack);
});
```

```
var http = require('http');

http.createServer(function(request, response) {
  var headers = request.headers;
  var method = request.method;
  var url = request.url;
  var body = [];
  request.on('error', function(err) {
    console.error(err);
  }).on('data', function(chunk) {
    body.push(chunk);
  }).on('end', function() {
    body = Buffer.concat(body).toString();
    // A este punto, tenemos las cabeceras, metodo, url y cuerpo y
    // podemos hacer lo necesario para responder a la solicitud.
  });
}).listen(8080); // Activamos el servidor, a la escucha del puerto 8080
```

```
var http = require('http');

http.createServer(function(request, response) {
  var headers = request.headers;
  var method = request.method;
  var url = request.url;
  var body = [];
  request.on('error', function(err) {
    console.error(err);
  }).on('data', function(chunk) {
    body.push(chunk);
  }).on('end', function() {
    body = Buffer.concat(body).toString();
    // A este punto, tenemos las cabeceras, metodo, url y cuerpo y
    // podemos hacer lo necesario para responder a la solicitud.
  });
}).listen(8080); // Activamos el servidor, a la escucha del puerto 8080
```

TIME OUT

TECSUP
Pasión por la Tecnología

Códigos de estado HTTP

- Los códigos de estado deberían ser por defecto 200. Nosotros podemos manipular esto enviando al navegador el código que deseemos.

```
response.statusCode = 404;
// Responder que no existe la solicitud.
```

Códigos de estado HTTP

- 1xx: estos códigos son solicitudes de información. Significa que se ha recibido la solicitud, con lo cual se puede continuar con el proceso.
- 2xx: se trata de códigos de éxito. Significa que la solicitud fue recibida con éxito y aceptada.
- 3xx: se trata de códigos de redirección. Significa que hay que hacer peticiones complementarias para completar la solicitud con éxito.
- 4xx: se trata de códigos de error y significa que la solicitud contiene una sintaxis incorrecta o no puede completarse.
- 5xx: se tratan de códigos de error del lado de servidor. Significa que el servidor está caído o simplemente no funcionó correctamente.

Responde headers

```
response.setHeader('Content-Type', 'application/json');
response.setHeader('X-Powered-By', 'tecsup');
```

Escribir explícitamente los headers

```
response.writeHead(200, {
   'Content-Type': 'application/json',
   'X-Powered-By': 'tecsup'
});
```

Response Body

- Como response es un WritableStream, puede ser manipulado para escribirle un Response Body.
- Así mismo, se puede pasar data opcional a la función end para enviar los últimos bits de data.
- Es importante establecer el código de estado y las cabeceras antes de empezar a escribir chunks de data en el response body.

Response Body

```
response.write('<html>');
response.write('<body>');
response.write('<h1>Hola Mundo!</h1>');
response.write('</body>');
response.write('</html>');
response.end();
```

```
response.end('<html><body><h1>Hola Mundo!</h1></body></html>');
```

GRACIAS POR SU ATENCIÓN

TECSUP
Pasión por la Tecnología