1. **Server HTTP Dasar**

Penggunaan Node.js yang revolusioner yaitu sebagai server. Yup...mungkin kita terbiasa memakai server seperti Apache - PHP, Nginx - PHP, Java - Tomcat - Apache atau IIS - ASP.NET sebagai pemroses data di sisi server, tetapi sekarang semua itu bisa tergantikan dengan memakai JavaScript - Node.js!. Lihat contoh dasar dari server Node.js berikut (kode sumber pada direktori code pada repositori ini).

server-http.js

```
var http = require('http'),
    PORT = 3400;

var server = http.createServer(function(req, res){
    var body = "<pre>Haruskah belajar Node.js?</pre><p><h3>...Yo
Mesto!</h3></p>"
    res.writeHead(200, {
        'Content-Length':body.length,
        'Content-Type':'text/html',
        'Pesan-Header':'Pengenalan Node.js'
    });

    res.write(body);
    res.end();
});

server.listen(PORT);

console.log("Port "+PORT+" : Node.js Server...");
```

Paket http merupakan paket bawaan dari platform Node.js yang mendukung penggunaan fitur-fitur protokol HTTP. Object `server` merupakan object yang di kembalikan dari fungsi `createServer()`.

```
var server = http.createServer([requestListener])
```

Tiap request yang terjadi akan ditangani oleh fungsi callback `requestListener`. Cara kerja callback ini hampir sama dengan ketika kita

menekan tombol button html yang mempunyai atribut event `onclick`, jika ditekan maka fungsi yang teregistrasi oleh event `onclick` yaitu `clickHandler(event)` akan dijalankan.

onclick-button.html

```
<script>
    function clickHandler(event){
        console.log(event.target.innerHTML+" Terus!");
    }
</script>

<button onclick="clickHandler(event)">TEKAN</button>
```

Sama halnya dengan callback `requestListener` pada object `server` ini jika ada request maka `requestListener` akan dijalankan

```
function(req, res){
    var body = "<pre>Haruskah belajar Node.js?</pre><p><h3>...Yo
Mesto!</h3></p>"
    res.writeHead(200, {
        'Content-Length':body.length,
        'Content-Type':'text/html',
        'Pesan-Header':'Pengenalan Node.js'
    });

    res.write(body);
    res.end();
}
```

Paket http Node.js memberikan keleluasan bagi developer untuk membangun server tingkat rendah. Bahkan mudah saja kalau harus men-setting nilai field header dari HTTP.

Seperti pada contoh diatas agar respon dari request diperlakukan sebagai HTML oleh browser maka nilai field `Content-Type` harus berupa `text/html`. Setting ini bisa dilakukan melalui metode `writeHead()`, `res.setHeader(field, value)` dan beberapa metode lainnya.

Untuk membaca header bisa dipakai fungsi seperti `res.getHeader(field, value)` dan untuk menghapus field header tertentu dengan memakai fungsi `res.removeHeader(field)`. Perlu diingat bahwa setting header dilakukan sebelum fungsi `res.write()` atau `res.end()` di jalankan karena jika `res.write()` dijalankan tetapi kemudian ada perubahan field header maka perubahan ini akan diabaikan.

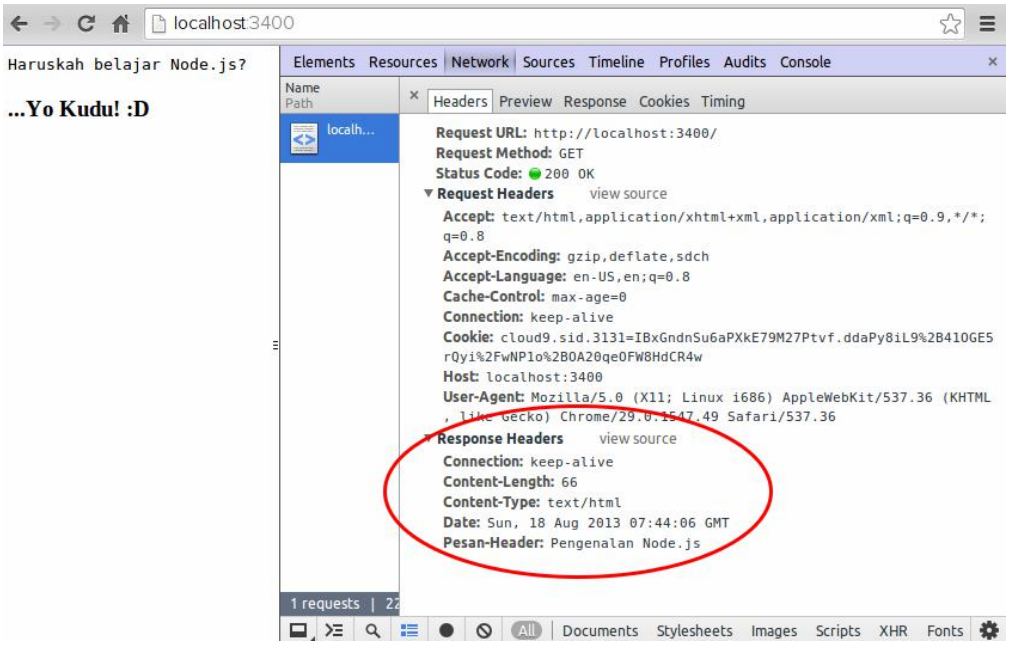
Satu hal lagi yaitu tentang kode status dari respon HTTP. Kode status ini bisa disetting selain 200 (request http sukses), misalnya bila diperlukan halaman error dengan kode status 404.

1.1. Menjalankan Server

Untuk menjalankan server Node.js ketik perintah berikut di terminal

```
$ node server-http.js
Port 3400 : Node.js Server...
```

Buka browser (chrome) dan buka url `http://localhost:3400` kemudian ketik `CTRL+SHIFT+I` untuk membuka Chrome Dev Tool, dengan tool ini bisa dilihat respon header dari HTTP dimana beberapa fieldnya telah diset sebelumnya (lingkaran merah pada screenshot dibawah ini).



2. Server File Statis

Aplikasi web memerlukan file - file statis seperti CSS, font dan gambar atau file - file library JavaScript agar aplikasi web bekerja sebagaimana mestinya. File - file ini sengaja dipisahkan agar terstruktur dan secara *best practices* file - file ini memang harus dipisahkan. Lalu bagaimana caranya Node.js bisa menyediakan file - file ini ?...ok mari kita buat server Node.js yang fungsinya untuk menyediakan file statis.

Agar server Node.js bisa mengirimkan file statis ke klien maka server perlu mengetahui `path` atau tempat dimana file tersebut berada.

Node.js bisa mengirimkan file tersebut secara streaming melalui fungsi `fs.createReadStream()`. Sebelum dijelaskan lebih lanjut mungkin bisa dilihat atau di coba saja server file Node.js dibawah ini

server-file.js

```
var http = require('http'),
    parse = require('url').parse,
    join = require('path').join,
    fs = require('fs'),
    root = join(__dirname, 'www'),
```

```
    PORT = 3300,

    server = http.createServer(function(req, res){
        var url = parse(req.url),
            path = join(root, url.pathname),
            stream = fs.createReadStream(path);

        stream.on('data', function(bagian){
            res.write(bagian);
        });

        stream.on('end', function(){
            res.end();
        });

        stream.on('error', function(){
            res.setHeader('Content-Type','text/html');

            var url_demo = "http://localhost:"+PORT+"/index.html";
            res.write("coba                buka                <a
href="+url_demo+">"+url_demo+"</a>");
            res.end();
        })

    });

    server.listen(PORT);
    console.log('Port '+PORT+': Server File ');
```

Berikut sedikit penjelasan dari kode diatas

- __dirname merupakan variabel global yang disediakan oleh Node.js yang berisi path direktori dari file yang sedang aktif mengeksekusi `__dirname`.
- `root` merupakan direktori root atau referensi tempat dimana file-file yang akan dikirimkan oleh server Node.js. Pada kode server diatas direktori root di setting pada direktori `www`.
- `path` adalah path file yang bisa didapatkan dengan menggabungkan path direktori root dan `pathname`. `pathname` yang dimaksud di sini misalnya jika URL yang diminta yaitu `http://localhost:3300/index.html`

maka `pathname` adalah `/index.html`. Nilai variabel `path` dihasilkan dengan memakai fungsi `join()`.

```
var path = join(root, url.pathname)
```

- `stream` yang di kembalikan oleh fungsi `fs.createReadStream()` merupakan class stream.Readable. Objek `stream` ini mengeluarkan data secara streaming untuk di olah lebih lanjut. Perlu menjadi catatan bahwa `stream.Readable` tidak akan mengeluarkan data jikalau tidak di kehendaki. Nah...cara untuk mendeteksi data streaming ini sudah siap di konsumsi atau belum adalah melalui event.

Event yang di dukung oleh class `stream.Readable` adalah sebagai berikut

- Event: `readable`
- Event: `data`
- Event: `end`
- Event: `error`
- Event: `close`

Mungkin anda bertanya kenapa server file statis diatas memakai stream, bukankah menyediakan file secara langsung saja sudah bisa? jawabannya memang bisa, tetapi mungkin tidak akan efisien kalau file yang akan di berikan ke client mempunyai ukuran yang besar. Coba lihat kode berikut

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

var server = http.createServer(function (req, res) {
    fs.readFile(__dirname + '/data.txt', function (err, data) {
        res.end(data);
    });
});
server.listen(8000);
```

Jika file `data.txt` terlalu besar maka buffer yang digunakan oleh sistem juga besar dan konsumsi memori juga akan bertambah besar seiring semakin banyak pengguna yang mengakses file ini.

Jika anda ingin lebih banyak mendalami tentang Node.js Stream silahkan lihat resource berikut (dalam Bahasa Inggris):

- Node.js API Stream
- Stream Handbook

Sumber: https://idjs.github.io/belajar-nodejs/