

NODE.JS BY EXAMPLES

FOR ANY OS



Daftar Isi

Source Code Penunjang Buku Ini.....	6
Bab 1. Mengenal Node.js.....	7
Apa Itu Node.js?.....	7
Apa yang Bisa Dilakukan dengan Node.js?.....	8
Bagaimana Cara Node.js Bekerja Mengangani Request?.....	8
Perbedaan Javascript Browser dengan Node.js.....	10
Mengapa Kita Menggunakan Node.js untuk Aplikasi Web?.....	10
Bab 2. Memulai Penggunaan Node.js.....	12
Meng-install Node.js di Windows.....	12
Meng-install Node.js di Linux (Ubuntu).....	13
Meng-install Node.js di MacOS.....	14
Menguji Hasil Peng-install-an Node.js.....	14
Membuat Project Node.js.....	16
Membuat Program Console "Hello World".....	17
Membuat Program Web "Hello World".....	18
Membuat Program REPL "Hello World".....	19
Bab 3. Referensi Singkat Javascript.....	21
Variabel dan Tipe Data.....	21
Konversi Tipe Data.....	22
Operator.....	23
Comparison.....	24
Conditional.....	25
Logical.....	26
Loop.....	27
Fungsi.....	29
Class.....	30
Promise.....	32
Bab 4. Modul-Modul Node.js.....	35
Beberapa Modul Bawaan.....	36
File System.....	36
HTTP.....	38
Events.....	39
Modul-Modul Bawaan Lainnya.....	42
Modul Luar.....	42
Modul Sendiri.....	44
Bab 5. Node.js Globals.....	45
__dirname.....	45
__filename.....	45
console.....	46
module.....	46
exports.....	46
global (tanpa "s").....	47

process.....	47
require.....	47
setTimeout.....	48
setInterval.....	48
setImmediate.....	48
Globals Lainnya.....	49
Bab 6. Express.js.....	56
Meng-install dan Menggunakan Express.js.....	56
Express.js Routing.....	58
Express.js Route Path.....	59
Response Methods.....	60
Middleware.....	61
Template Engine.....	64
Menggunakan Template Engine EJS.....	64
Bab 7. Database.....	67
MySQL.....	67
Meng-install MySQL.....	68
Membuka Koneksi ke MySQL.....	68
Fungsi Query pada Node.js MySQL.....	69
SQLite.....	72
Meng-install SQLite.....	72
Fungsi Run dan Prepare pada Node SQLite3.....	72
Fungsi Each.....	73
Bab 8. Websocket.....	75
Apa Itu Websocket?.....	75
Apa Bedanya dengan AJAX?.....	76
Membuat Aplikasi Websocket Pertama.....	77
Event-Event pada Websocket.....	79
Broadcast Data dengan Websocket.....	85
Mendapatkan IP Address Client dengan Websocket.....	90
Menggunakan Socket.io dengan Express.js.....	92
Bab 9. Debugging.....	95
Debugging dengan PowerShell/Terminal.....	96
Debugging dengan Visual Studio Code.....	98
Bab 10. Unit Testing.....	104
Bab 11. Deployment.....	111
Cara Men-Deploy Program Node.js di Heroku.....	111
Cara Men-Deploy Program Node.js di Ubuntu Server.....	114
Cara Men-Deploy Banyak Program Node.js dalam satu Ubuntu Server	
Menggunakan Apache.....	116
Bab 12. C++ Addons.....	118
Membuat Program Hello World C++ di Windows.....	119
Persiapan.....	119
Menulis Kode Program Hello World.....	120

Membuat Program Hello World C++ di Ubuntu.....	123
Persiapan.....	123
Menulis Kode Program Hello World.....	124
Bab 13. Praktikum: Membuat Website Clone Pastebin.....	126
Yang Anda Butuhkan sebelum Kita Belajar.....	129
Membedah Aplikasi Pastebin.....	131
Halaman Utama (New Paste).....	131
Halaman Trends.....	133
Halaman Search.....	134
API, Tools, FAQ dan Sisanya.....	135
Merancang Node.js Pasta.....	135
Rancangan User Interface.....	136
Package NPM yang Dibutuhkan.....	138
Package Non-NPM yang Dibutuhkan.....	139
Memprogram Node.js Pasta.....	140
Struktur Direktori Project.....	140
Menempatkan Package-Package Non NPM di Direktori Project.....	141
Menginstall Package NPM di Direktori Project.....	141
Mengimpor Package NPM.....	143
Mempersiapkan Framework Expressjs.....	144
Membuat Database dengan SQLite3.....	148
Membuat Fungsi-Fungsi Utility.....	152
Membuat Halaman 404 Not Found.....	156
Membuat Halaman Home (New Paste).....	159
Membuat Halaman Manage.....	165
Membuat Halaman Paste.....	175
Membuat Halaman Trends.....	184
Mencoba Aplikasi Ini.....	188
Men-Deploy Aplikasi Ini di Heroku.....	189
Bab 14. Praktikum: Membuat Web System Monitor.....	191
Apa yang Akan Kita Buat.....	191
Mengapa Kita Membuatnya.....	193
Bagaimana Kita Membuatnya.....	194
Yang Anda Butuhkan sebelum Kita Belajar.....	195
Menginstall Node.js dan NPM.....	195
Membuat Project Node.js.....	196
Menginstall Package yang Dibutuhkan.....	197
Vue.js.....	197
Vue-Resource.....	198
Bootstrap.....	198
jQuery.....	198
Chart.js.....	198
OS-Utils Module.....	198
Expressjs.....	198

FS Module.....	199
Mengimpor Semua Package yang Telah Kita Install.....	199
Lalu bagaimana dengan Vue.js dan Chart.js?.....	200
Membuat Kerangka Request Handler.....	200
Mengetes Request Handler Express.....	200
Mengembangkan Request Handler Express Pada Aplikasi Kita.....	201
Membuat Pilihan Port yang Didengar Lebih Dinamis.....	204
Merancang User Interface.....	206
Memberi Akses File Statis Dari Server.....	206
Mencoba Mengakses File Statis dari Server.....	208
Merespon Setiap Request dalam Bentuk JSON dari Server.....	211
Mengimplementasikan Client Side Script pada File Index.html.....	216
Penutup.....	233
Daftar Pustaka.....	234
Lampiran.....	236

Source Code Penunjang Buku Ini

Source code buku ini dan versi PDF dari buku ini bisa didapatkan di bab Lampiran.

Bab 1. Mengenal Node.js

Aplikasi web adalah salah satu software aplikasi yang paling beragam di dunia.

Untuk membuatnya, dibutuhkan beberapa skill sekaligus.

Untuk tampilannya saja, dibutuhkan skill Javascript, HTML, dan CSS.

Selain itu, agar konten dari aplikasi web bisa dinamis, diperlukan server side scripting seperti PHP.

Jadi, kalau dijumlahkan setidaknya diperlukan 4 skill sekaligus untuk membuat sebuah aplikasi web seorang diri.

Walaupun demikian, saat ini ada sebuah teknologi baru berupa runtime environment yang menggunakan Javascript sebagai bahasa scripting-nya.

Teknologi tersebut adalah Node.js.

Akibat dari munculnya Node.js, maka scripting di client side dan server side bisa menggunakan bahasa yang sama, yakni Javascript.

Apa Itu Node.js?

Node.js merupakan sebuah runtime environment berbasis Javascript yang bersifat cross platform dan open source.

Node.js bisa digunakan untuk membuat sebuah aplikasi server untuk web.

Walaupun begitu, Node.js juga sebenarnya bisa juga digunakan untuk membuat aplikasi console yang berjalan di command line interface seperti PowerShell atau Terminal.

Jenis aplikasi web yang bisa dibuat juga bisa bermacam-macam.

Mulai dari aplikasi blogging, social networking, payment gateway, search engine, web scraper, dan lain-lain.

Dengan lisensinya yang sangat permisif, Node.js bisa didapatkan dengan gratis.

Node.js juga dapat berjalan di banyak Operating System seperti Windows, Linux, Unix, dan Mac OS sehingga Node.js bisa dikatakan cross platform.

Apa yang Bisa Dilakukan dengan Node.js?

Hal yang paling umum dalam aplikasi web yang bisa dilakukan oleh Node.js adalah memanipulasi file.

File dalam artian file system maupun file dalam artian database file.

Jadi, kita bisa membuka, menutup, mengedit, dan menghapus file dalam file system dan juga melakukan fungsi-fungsi yang berkaitan dengan database (meskipun diperlukan modul luar untuk database).

Node.js juga bisa membuat respon ke client berupa HTML, JSON, dan sebagainya dari server.

Artinya, Node.js bisa digunakan untuk membuat konten yang dinamis.

Selain itu, Node.js juga bisa menyaring request dan parameter-parameternya, yang artinya Node.js bisa menangani form HTML dari client ke server.

Node.js juga dilengkapi dengan NPM, yakni sebuah package manager untuk mendapatkan modul-modul luar.

Dengan menggunakan NPM, struktur project Node.js jadi lebih baik dan terorganisir.

Modul-modul luar tersebut sangat banyak dan bisa didapatkan dari www.npmjs.com.

Akan tetapi, ada juga modul-modul tertentu yang hanya bisa dijalankan pada OS tertentu.

Jadi, berhati-hatilah ketika memilih modul yang akan digunakan untuk berbagai OS.

Pastikan modul tersebut benar-benar bisa dijalankan pada OS yang ditargetkan.

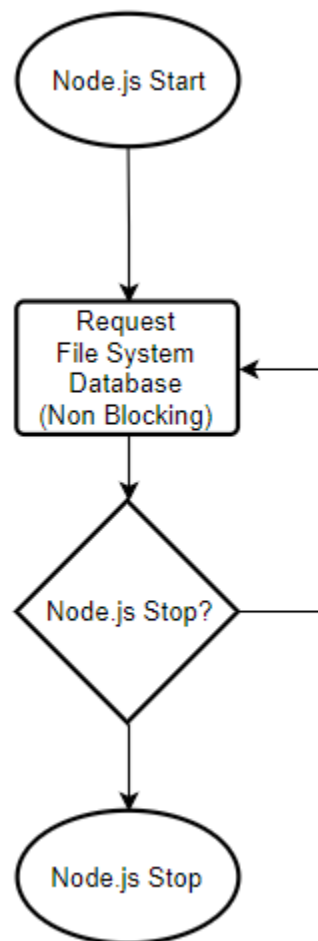
Bagaimana Cara Node.js Bekerja Mengangani Request?

Node.js merespon sebuah request dengan cara yang berbeda dengan alternatif lainnya.

Ketika Node.js menerima perintah dari client di server, Node.js akan segera bersiap menangani request selanjutnya.

Ketika ada file yang diminta untuk dibaca di server, Node.js membuka file tersebut dan secara bersamaan mengembalikan konten yang sudah diproses sebelumnya ke client.

Jadi, Node.js tidak menunggu-nunggu untuk menangani request antara yang sekarang, dengan yang selanjutnya.



Node.js Process

Hal tersebut berbeda dengan alternatif lainnya.

PHP misalnya, dia akan tetap menunggu sampai file yang terbuka selesai dibaca, baru kemudian menangani request selanjutnya.

Perbedaan Javascript Browser dengan Node.js

Walaupun sama-sama Javascript, Javascript-nya Node.js berbeda dengan Javascript-nya browser. Berikut ini perbedaannya.

	Javascript Browser	Javascript Node.js
Window Object	Ada	Tidak Ada
Location Object	Ada	Tidak Ada
Document Object	Ada	Tidak Ada
Require	Tidak Ada	Ada
Memproses Apa	Response	Request

Mengapa Kita Menggunakan Node.js untuk Aplikasi Web?

Node.js merupakan runtime environment yang penggunaan memorinya sangat efisien karena menggunakan single thread.

Berbeda dengan PHP yang menggunakan multithread.

Dengan demikian, Node.js cocok untuk aplikasi web yang menangani banyak request sekaligus seperti aplikasi chatting.

Selain itu, Node.js menggunakan bahasa Javascript.

Artinya, kita bisa menyeragamkan bahasa di sisi client maupun server, sehingga lebih hemat waktu belajar.

Node.js juga memiliki modul yang berlimpah, baik yang bawaan maupun yang luaran dari www.npmjs.com.

Semua modul luaran itu bisa didapatkan dengan menggunakan NPM.

Dengan banyaknya modul tersebut, kita bisa fokus pada fungsi utama aplikasi web yang kita buat.

Di samping itu, komunitas pengguna Node.js sangat banyak dan aktif.

Apabila kita mendapatkan masalah dalam pengembangan aplikasi dengan Node.js, kita bisa googling solusinya dengan mudah.

Dan yang paling menarik, beberapa perusahaan IT besar telah mengadopsi sistem Node.js untuk website-nya dan itu bisa menjadi teladan bagi mereka yang masih ragu untuk menggunakan Node.js.

Beberapa perusahaan IT besar yang menggunakan Node.js adalah:

- Paypal
- LinkedIn
- Yahoo
- Mozilla

Bab 2. Memulai Penggunaan Node.js

Pada bab sebelumnya, kita telah membahas sedikit tentang Node.js.

Agar lebih jelas, sekarang kita mencoba menggunakan Node.js untuk membuat aplikasi yang sangat sederhana.

Tapi sebelumnya, ada beberapa hal yang dibutuhkan.

Hal-hal tersebut adalah:

- Text Editor apapun, boleh Notepad++ (Windows Saja) atau Visual Studio Code atau Atom Editor. Disarankan untuk menggunakan Visual Studio Code karena akan dibahas cara debugging Node.js dengan Visual Studio Code3.
- PowerShell (untuk Windows, sudah ada) atau Terminal (untuk Linux atau Mac, sudah ada)
- Koneksi Internet
- Node.js yang bisa didapatkan di <http://nodejs.org>

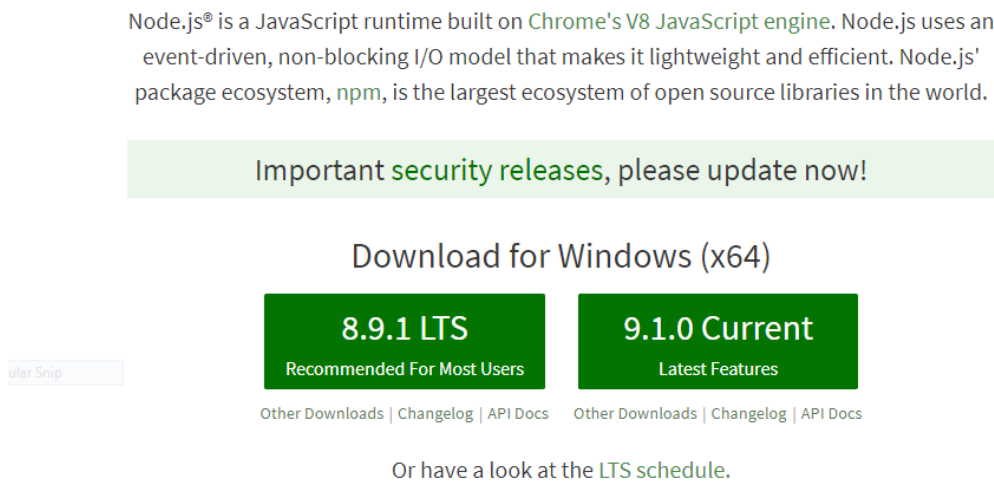
Khusus Node.js, saya akan membahas cara meng-install-nya pada OS yang berbeda.

Meng-install Node.js di Windows

Untuk meng-install Node.js di Windows, kita perlu memiliki file installer yang telah didapatkan dari <http://nodejs.org>

Setelah membuka situs tersebut, pilih Node.js yang "Recommended For Most Users".

Pada gambar ini ada di tombol hijau sebelah kiri.



Download Node.js, Pilih yang Kiri

Setelah installer Node.js selesai di-download, kita tinggal menjalankan installer-nya dan klik next sampai finish.

Meng-install Node.js di Linux (Ubuntu)

Meng-install Node.js di Linux sedikit lebih susah daripada di Windows.

Karena distro Linux sangat beragam, saya memilih yang paling umum digunakan saja, yakni Ubuntu.

Berbeda dengan proses peng-install-an di Windows, kita meng-install Node.js di Ubuntu melalui repository Ubuntu.

Jadi, kita akan menggunakan Terminal untuk melakukannya.

Pertama-tama, buka terminal, pastikan kita berada pada directory "home", lalu ketik ini untuk meng-install curl:

```
sudo apt-get install curl
```

Kita perlu meng-install curl karena kita akan meng-install installation script-nya terlebih dahulu.

Untuk meng-install installation script-nya:

```
curl -sL https://deb.nodesource.com/setup_6.x -o nodesource_setup.sh
```

Lantas, jalankan script tersebut dengan cara:

```
sudo bash nodesource_setup.sh
```

Selanjutnya, install Node.js:

```
sudo apt-get install nodejs
```

Install juga build-essential karena kemungkinan ada package NPM yang membutuhkannya:

```
sudo apt-get install build-essential
```

Meng-install Node.js di MacOS

Untuk meng-install Node.js di MacOS, caranya mirip dengan di Windows.

Download installer Node.js di website resminya dan pilih installer untuk MacOS. Selanjutnya, jalankan installer tersebut di MacOS dan ikuti instruksinya hingga selesai.

Menguji Hasil Peng-install-an Node.js

Agar kita yakin bahwa Node.js telah terinstall dengan benar, kita akan mengujinya terlebih dahulu.

Buka PowerShell/Terminal kemudian ketikkan:

```
npm -v
```

Jika hasilnya semacam ini:

```
5.5.1
```

Berarti NPM baik-baik saja.

Selanjutnya ketik ini:

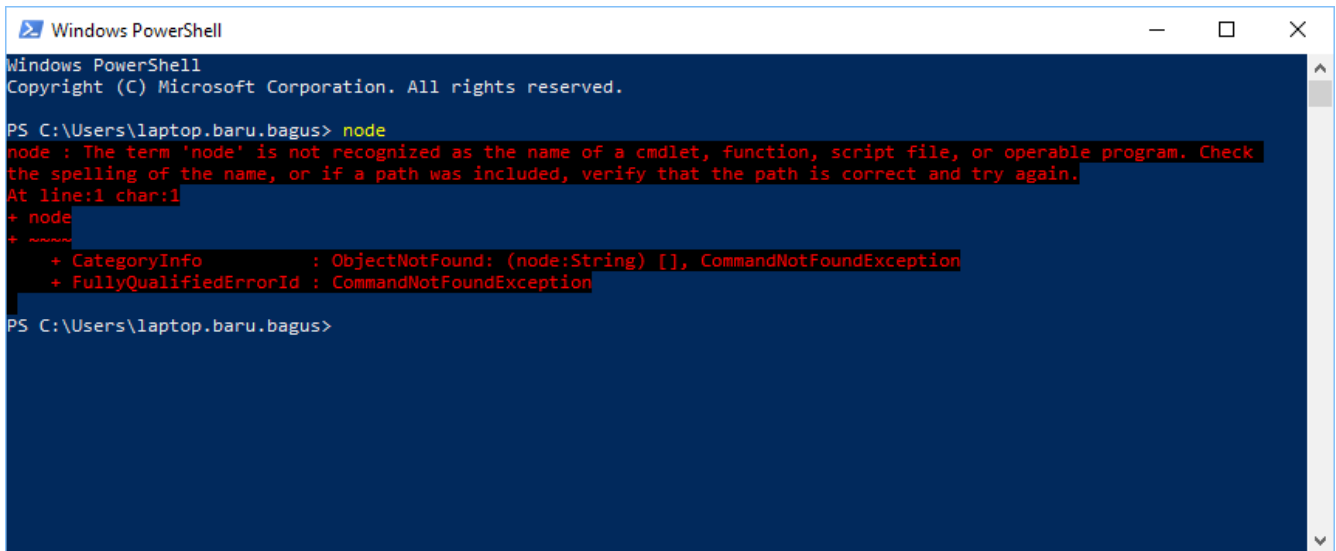
```
node -v
```

Jika hasilnya seperti ini:

v8.9.0

Berarti Node.js baik-baik saja.

Apabila hasil-hasil perintah tadi tidak keluar atau ada pesan semacam ini:

A screenshot of a Windows PowerShell window. The title bar says "Windows PowerShell". The text inside shows the command prompt at "PS C:\Users\laptop.baru.bagus>". The user has entered the command "node". The output is an error message in red text: "node : The term 'node' is not recognized as the name of a cmdlet, function, script file, or operable program. Check the spelling of the name, or if a path was included, verify that the path is correct and try again. At line:1 char:1". Below this, there is a detailed error object: "+ node", "+ CategoryInfo : ObjectNotFound: (node:String) [], CommandNotFoundException", and "+ FullyQualifiedErrorId : CommandNotFoundException". The prompt returns to "PS C:\Users\laptop.baru.bagus>".

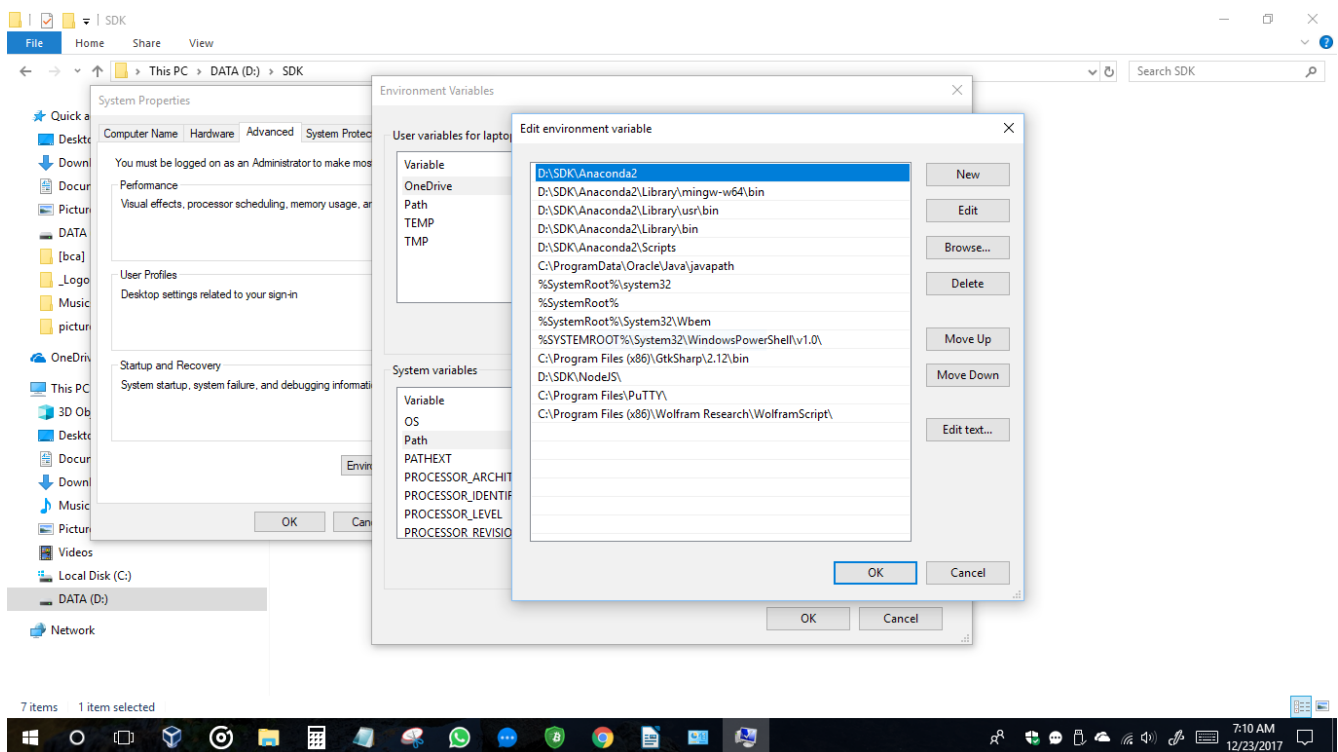
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Users\laptop.baru.bagus> node
node : The term 'node' is not recognized as the name of a cmdlet, function, script file, or operable program. Check
the spelling of the name, or if a path was included, verify that the path is correct and try again.
At line:1 char:1
+ node
+ ~~~~~
+ CategoryInfo          : ObjectNotFound: (node:String) [], CommandNotFoundException
+ FullyQualifiedErrorId : CommandNotFoundException

PS C:\Users\laptop.baru.bagus>
```

Berarti ada yang salah dengan setting Environment Variable (jika di Windows).

Cara memperbaikinya adalah dengan menge-set Environment Variable "PATH" ke lokasi NPM atau Node.js ter-install.



Seperti gambar di atas.

Membuat Project Node.js

Sebelum kita mulai mencoba coding dengan Node.js, dianjurkan untuk membuat project dahulu dengan NPM.

Langkah ini tidak wajib, memang, tapi dengan membuat project dengan NPM, maka file "package.json" akan dibuat dan itu akan memudahkan kita dalam melakukan deployment.

Untuk membuat project, pertama-tama buat folder bernama "bab-2" di dalam folder apapun.

Kemudian buka PowerShell/Terminal di folder tersebut.

Selanjutnya, ketikkan perintah:

```
npm init
```

Selanjutnya isi semua input yang diminta hingga selesai.

Setelah input selesai dimasukkan, maka akan muncul file "package.json".

Isi dari file "package.json" seperti ini:

```
{
  "name": "bab-2",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Anonim",
  "license": "UNLICENSED"
}
```

Sejak saat itu, kita bebas menambah file-file Node.js apapun di dalam root folder yang sama dengan file "package.json".

Tapi sekali lagi, hal tersebut tidak wajib jika kita hanya ingin menjalankan program Node.js di PC kita.

Membuat Program Console "Hello World"

Sekarang, saatnya mencoba coding dengan Node.js.

Kita akan membuat program "Hello World" yang akan menampilkan teks "Hello World" di PowerShell/Terminal kita.

Caranya, buat file baru bernama "bab-2-console-hello-world.js" pada folder "bab-2".

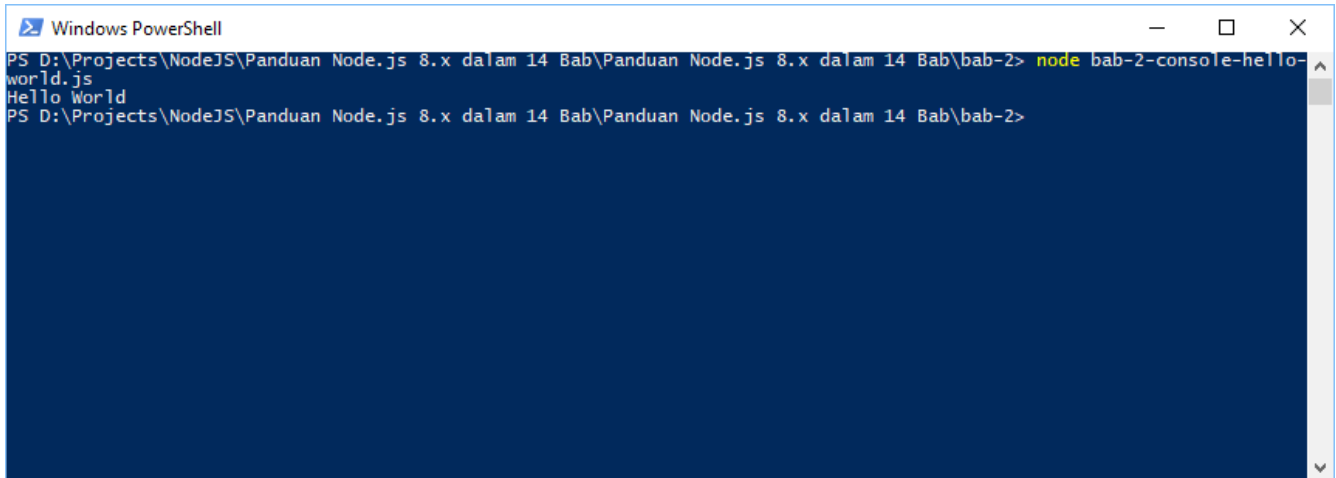
Kemudian isi file tersebut dengan kode ini:

```
console.log("Hello World");
```

Simpan file tersebut, kemudian buka PowerShell/Terminal pada folder "bab-2" dan jalankan perintah ini:

```
node bab-2-console-hello-world.js
```

Outputnya adalah:



```
Windows PowerShell
PS D:\Projects\NodeJS\Panduan Node.js 8.x dalam 14 Bab\Panduan Node.js 8.x dalam 14 Bab\bab-2> node bab-2-console-hello-world.js
Hello World
PS D:\Projects\NodeJS\Panduan Node.js 8.x dalam 14 Bab\Panduan Node.js 8.x dalam 14 Bab\bab-2>
```

Jadi, fungsi `console.log` adalah untuk mem-print teks yang di-input-kan ke dalamnya.

Membuat Program Web "Hello World"

Kali ini kita akan membuat program yang seperti tadi, tapi teksnya akan muncul di browser kita.

Pertama, buatlah file baru bernama "bab-2-web-hello-world.js" pada folder "bab-2".

Kemudian, isi file tersebut dengan kode ini:

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

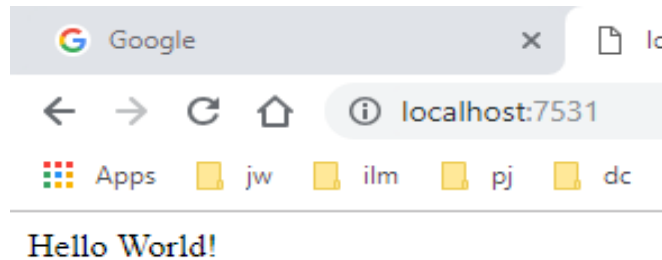
http.createServer(function (req, res) {
  let ct = 'text/html';
  let ct1 = 'Content-Type';
  let responseCode = 200;
  res.writeHead(responseCode, {ct1: ct});
  res.end('Hello World!');
}).listen(7531);
```

Jalankan dengan cara membuka PowerShell/Terminal pada folder "bab-2", lalu ketikkan perintah:

```
node bab-2-web-hello-world.js
```

Setelah itu, buka browser kita di `http://localhost:7531`

Nanti di browser, akan tampil teks ini:



Kode ini membuat server HTTP:

```
http.createServer(function (req, res) {
```

Request ada di argument "req" dan response-nya ditulis dari argument "res".

Bagian ini menulis header HTTP yang berupa "text/html":

```
res.writeHead(responseCode, {ct1: ct});
```

Sedangkan yang ini mengakhirinya dengan menampilkan teks "Hello World":

```
res.end('Hello World!');
```

Membuat Program REPL "Hello World"

REPL adalah singkatan dari Read, Eval, Print, dan Loop.

Dengan ini kita bisa mencoba kode Node.js tanpa menyimpannya dalam file script.

Untuk membuat "Hello World" dengan REPL, buka PowerShell/Terminal, kemudian jalankan perintah ini:

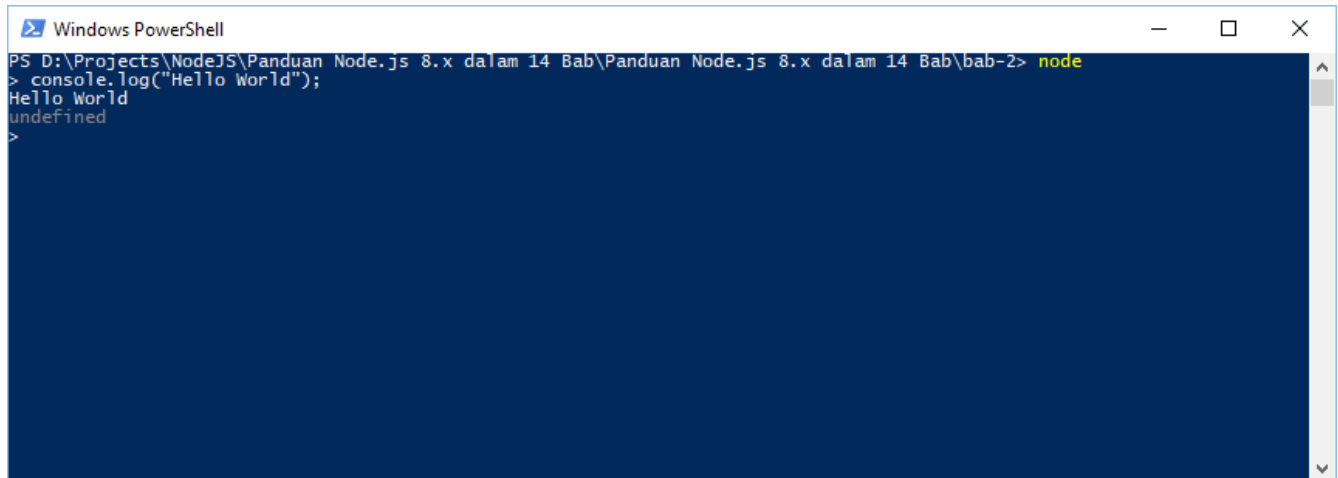
```
node
```

Selanjutnya, tuliskan kode ini:

```
console.log("Hello World");
```

Lalu tekan enter.

Nanti outputnya akan muncul di PowerShell/Terminal seperti ini:



```
Windows PowerShell
PS D:\Projects\NodeJS\Panduan Node.js 8.x dalam 14 Bab\Panduan Node.js 8.x dalam 14 Bab\bab-2> node
> console.log("Hello World");
Hello World
undefined
>
```

Bab 3. Referensi Singkat Javascript

Javascript adalah bahasa yang digunakan dalam Node.js.

Oleh karena itu, jika Anda baru pertama kali menggunakan Javascript, sebaiknya baca juga bab ini.

Variabel dan Tipe Data

Untuk mendeklarasikan variabel di Javascript, gunakan keyword "let" atau "var" atau "const".

```
let ipAddress = '127.0.0.1';
let port = 21;

var name = "Anonim";
var age = 1000;

const materi = "node.js";
const pi = 3.14;

console.log(ipAddress + " bertipe " + typeof(ipAddress));
console.log(port + " bertipe " + typeof(port));
console.log(name + " bertipe " + typeof(name));
console.log(age + " bertipe " + typeof(age));
console.log(materi + " bertipe " + typeof(materi));
console.log(pi + " bertipe " + typeof(pi));

//hapus '//' di bawah ini akan membuat error karena pi konstan
// pi = pi + 1;

console.log(pi);
```

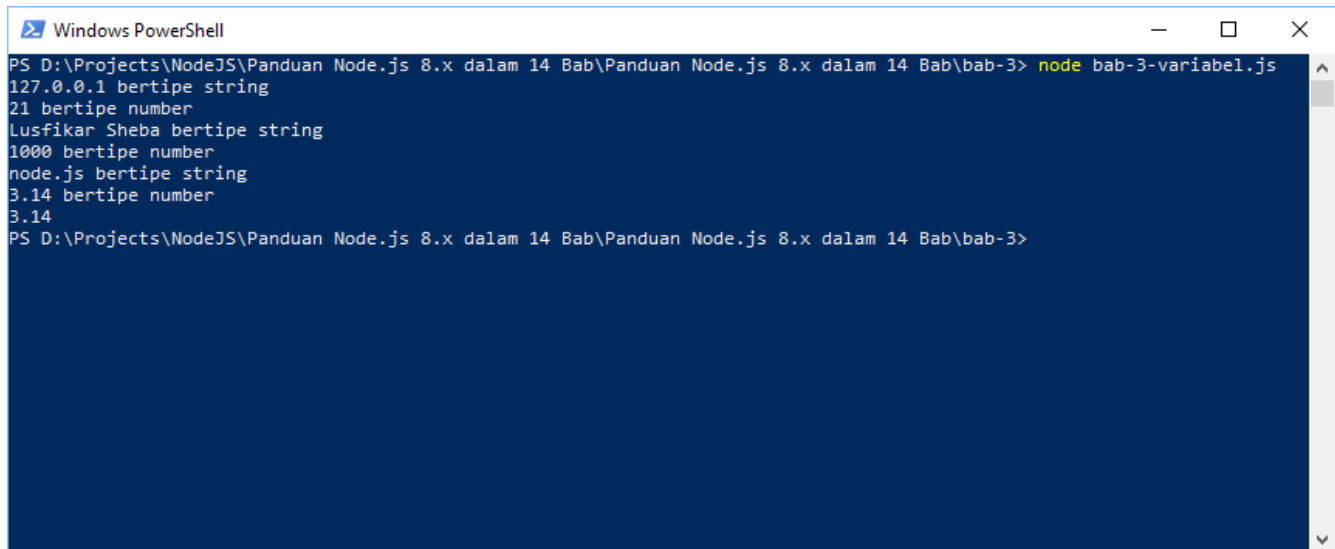
Untuk mencobanya, buat folder baru bernama "bab-3", kemudian buat file baru di dalam folder tersebut dengan nama "bab-3-variabel.js".

Isi file tadi dengan kode di atas.

Kemudian, buka PowerShell/Terminal pada folder bab-3 dan jalankan perintah ini:

```
node bab-3-variabel.js
```

Hasilnya seperti ini:



```
Windows PowerShell
PS D:\Projects\NodeJS\Panduan Node.js 8.x dalam 14 Bab\Panduan Node.js 8.x dalam 14 Bab\bab-3> node bab-3-variabel.js
127.0.0.1 bertipe string
21 bertipe number
Lusfika Sheba bertipe string
1000 bertipe number
node.js bertipe string
3.14 bertipe number
3.14
PS D:\Projects\NodeJS\Panduan Node.js 8.x dalam 14 Bab\Panduan Node.js 8.x dalam 14 Bab\bab-3>
```

Konversi Tipe Data

Variabel Javascript bisa dikonversi dari suatu tipe ke tipe yang lain:

```
var suatuInteger = 1985;
console.log(suatuInteger + " bertipe " + typeof(suatuInteger));

var jadiString = suatuInteger.toString();
console.log(jadiString + " bertipe " + typeof(jadiString));

var stringJadiNumber = Number(jadiString);
console.log(stringJadiNumber + " bertipe " + typeof(stringJadiNumber));

var benarSalah = true;
console.log(benarSalah + " bertipe " + typeof(benarSalah));

var booleanJadiString = benarSalah.toString();
console.log(booleanJadiString + " bertipe " + typeof(booleanJadiString));

var stringJadiBoolean = Boolean(booleanJadiString);
console.log(stringJadiBoolean + " bertipe " + typeof(stringJadiBoolean));
```

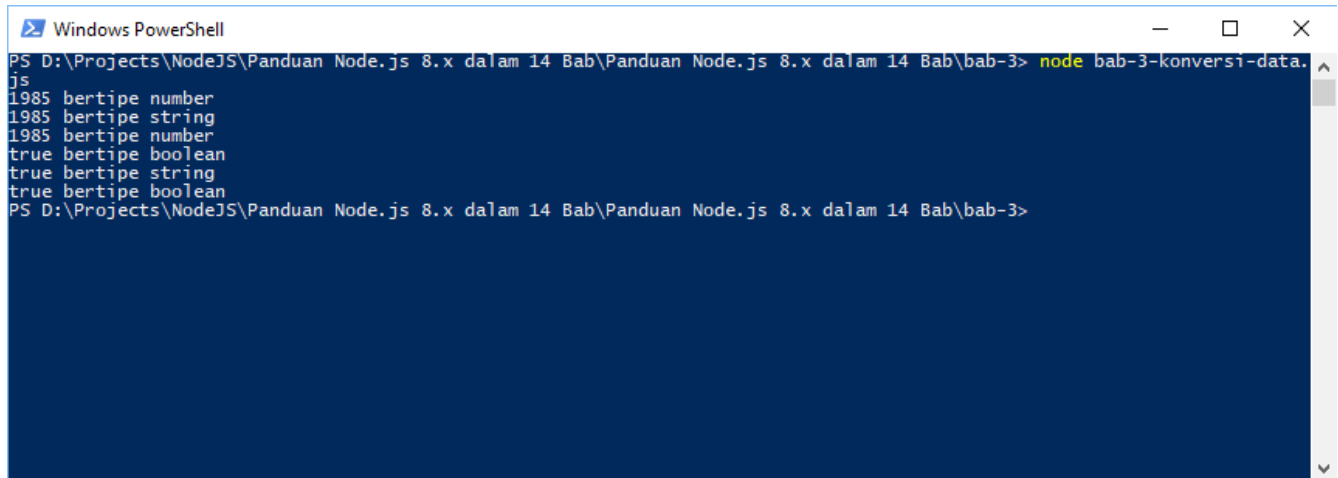
Untuk mencobanya, masuk ke folder bernama "bab-3", kemudian buat file baru di dalam folder tersebut dengan nama "bab-3-konversi-data.js".

Isi file tadi dengan kode di atas.

Kemudian, buka PowerShell/Terminal pada folder bab-3 dan jalankan perintah ini:

`node bab-3-konversi-data.js`

Hasilnya seperti ini:

A screenshot of a Windows PowerShell terminal window. The title bar says "Windows PowerShell". The command prompt shows the path "PS D:\Projects\NodeJS\Panduan Node.js 8.x dalam 14 Bab\Panduan Node.js 8.x dalam 14 Bab\bab-3>" followed by the command "node bab-3-konversi-data.js". The output of the command is displayed on the next line: "1985 bertipe number", "1985 bertipe string", "1985 bertipe number", "true bertipe boolean", "true bertipe string", and "true bertipe boolean". The prompt then returns to "PS D:\Projects\NodeJS\Panduan Node.js 8.x dalam 14 Bab\Panduan Node.js 8.x dalam 14 Bab\bab-3>".

```
PS D:\Projects\NodeJS\Panduan Node.js 8.x dalam 14 Bab\Panduan Node.js 8.x dalam 14 Bab\bab-3> node bab-3-konversi-data.js
1985 bertipe number
1985 bertipe string
1985 bertipe number
true bertipe boolean
true bertipe string
true bertipe boolean
PS D:\Projects\NodeJS\Panduan Node.js 8.x dalam 14 Bab\Panduan Node.js 8.x dalam 14 Bab\bab-3>
```

Operator

Javascript bisa menggunakan operasi matematika:

```
console.log(1+1); //2
console.log(1-1); //0
console.log(2*10); //20

var a = 5;
console.log(a ** 2); //a pangkat 2
console.log(a % 2); //a mod 2
a++; //a = a + 1;
console.log(a);
```

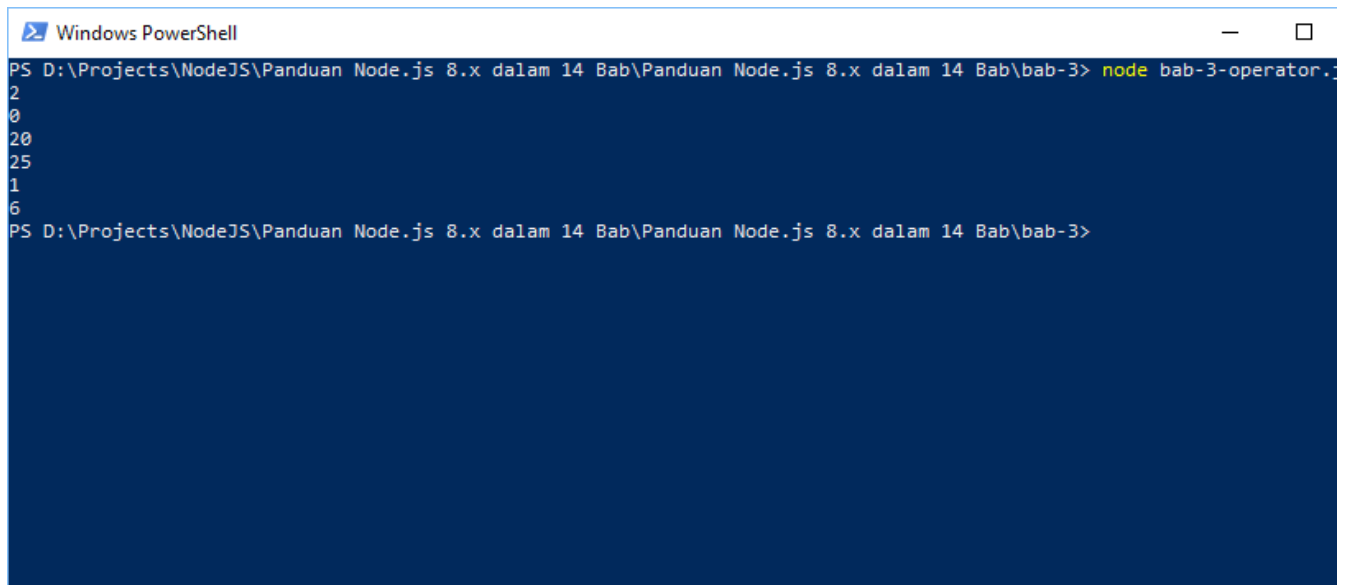
Untuk mencobanya, masuk ke folder bernama "bab-3", kemudian buat file baru di dalam folder tersebut dengan nama "bab-3-operator.js".

Isi file tadi dengan kode di atas.

Kemudian, buka PowerShell/Terminal pada folder bab-3 dan jalankan perintah ini:

```
node bab-3-operator.js
```

Hasilnya seperti ini:



```
Windows PowerShell
PS D:\Projects\NodeJS\Panduan Node.js 8.x dalam 14 Bab\Panduan Node.js 8.x dalam 14 Bab\bab-3> node bab-3-operator.js
2
0
20
25
1
6
PS D:\Projects\NodeJS\Panduan Node.js 8.x dalam 14 Bab\Panduan Node.js 8.x dalam 14 Bab\bab-3>
```

Comparison

Javascript bisa melakukan comparison:

```
console.log(2 > 1); //true
console.log(2 == 1); //false
console.log(2 != 1); //true
console.log(0 == false); //true
console.log('' == false); //true
console.log(0 === false); //false
console.log(null === undefined); //false
console.log(null == undefined); //true
console.log(null > 0); //false
console.log(null == 0); //false
console.log(null >= 0); //true
console.log(undefined > 0); //false
console.log(undefined < 0); //false
console.log(undefined == 0); //false
```

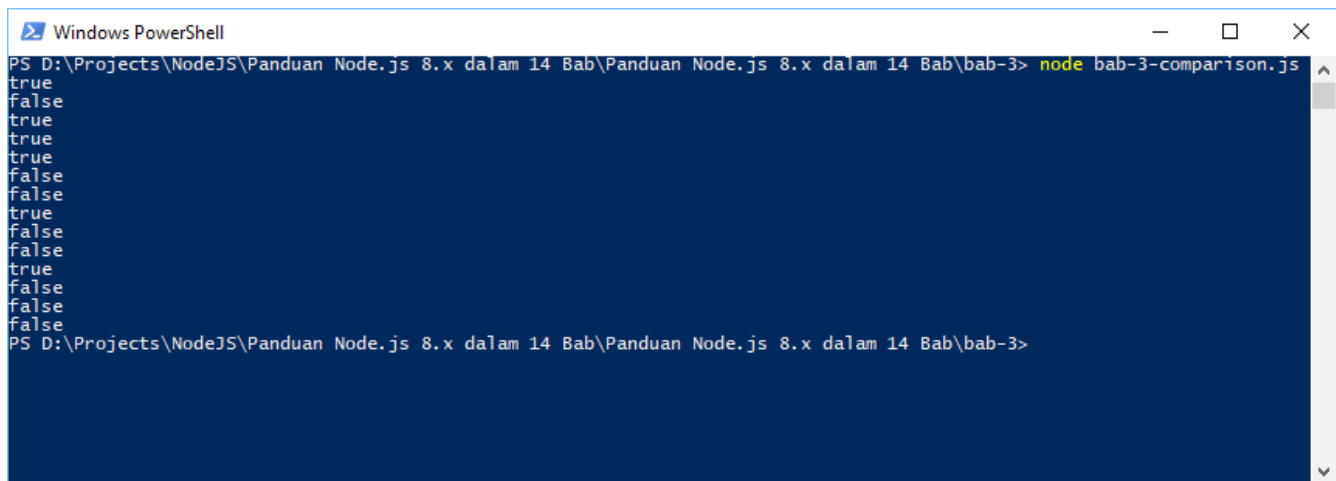

Untuk mencobanya, masuk ke folder bernama "bab-3", kemudian buat file baru di dalam folder tersebut dengan nama "bab-3-comparison.js".

Isi file tadi dengan kode di atas.

Kemudian, buka PowerShell/Terminal pada folder bab-3 dan jalankan perintah ini:

```
node bab-3-comparison.js
```

Hasilnya seperti ini:



```
Windows PowerShell
PS D:\Projects\NodeJS\Panduan Node.js 8.x dalam 14 Bab\Panduan Node.js 8.x dalam 14 Bab\bab-3> node bab-3-comparison.js
true
false
true
true
true
false
false
true
false
false
true
false
false
false
PS D:\Projects\NodeJS\Panduan Node.js 8.x dalam 14 Bab\Panduan Node.js 8.x dalam 14 Bab\bab-3>
```

Comparison ini nantinya bisa dimasukkan ke conditional seperti "if", "else", dan "else if".

Conditional

Contoh dari conditional adalah:

```
var a = 5;

if(a == 5){
  console.log("a == 5");
} else if(a == 6){
  console.log("a == 6");
} else if(a == 7){
  console.log("a == 7");
} else {
  console.log("a bukan 5, 6, dan 7");
}
```

```
var str = a == 5 ? "lima" : "bukan lima";  
console.log(str); //lima
```

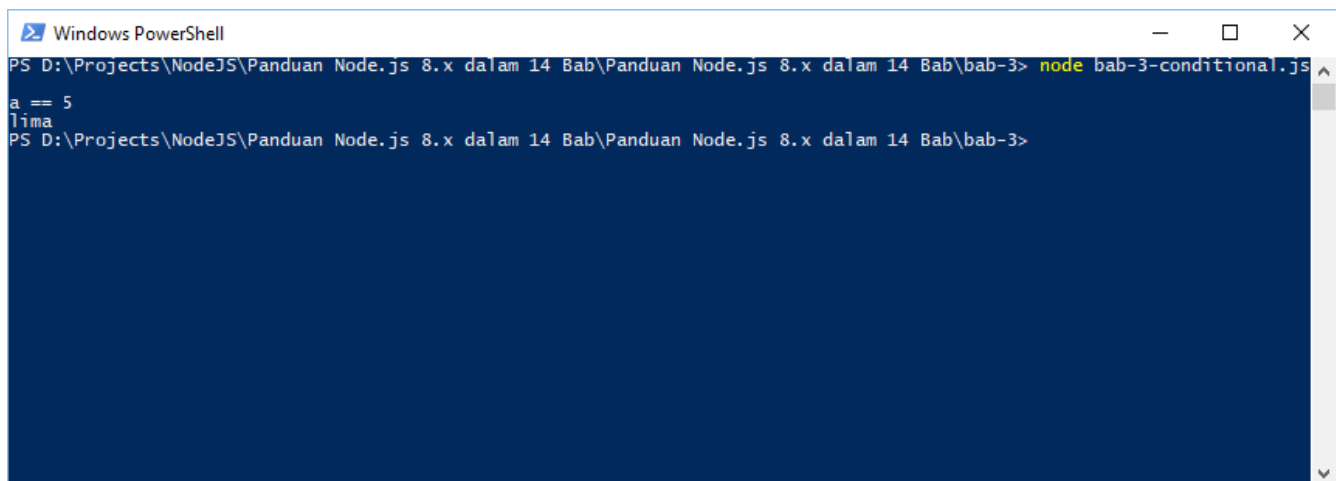
Untuk mencobanya, masuk ke folder bernama "bab-3", kemudian buat file baru di dalam folder tersebut dengan nama "bab-3-conditional.js".

Isi file tadi dengan kode di atas.

Kemudian, buka PowerShell/Terminal pada folder bab-3 dan jalankan perintah ini:

```
node bab-3-conditional.js
```

Hasilnya seperti ini:

A screenshot of a Windows PowerShell terminal window. The title bar reads "Windows PowerShell". The command prompt shows the path "PS D:\Projects\NodeJS\Panduan Node.js 8.x dalam 14 Bab\Panduan Node.js 8.x dalam 14 Bab\bab-3>" followed by the command "node bab-3-conditional.js". The output of the script is displayed on the next line: "a == 5" followed by "lima" on the next line. The prompt then returns to "PS D:\Projects\NodeJS\Panduan Node.js 8.x dalam 14 Bab\Panduan Node.js 8.x dalam 14 Bab\bab-3>".

```
Windows PowerShell  
PS D:\Projects\NodeJS\Panduan Node.js 8.x dalam 14 Bab\Panduan Node.js 8.x dalam 14 Bab\bab-3> node bab-3-conditional.js  
a == 5  
lima  
PS D:\Projects\NodeJS\Panduan Node.js 8.x dalam 14 Bab\Panduan Node.js 8.x dalam 14 Bab\bab-3>
```

Silakan coba ganti nilai a dengan yang lain dan lihat output-nya.

Logical

Javascript bisa mengerjakan operator logika.

OR dilambangkan dengan "||" dan AND dilambangkan dengan "&&":

```
console.log(true || true); //true  
console.log(false || true); //true  
console.log(true || false); //true  
console.log(false || false); //false  
  
console.log(true && true); //true
```

```
console.log(false && true); //false
console.log(true && false); //false
console.log(false && false); //false
```

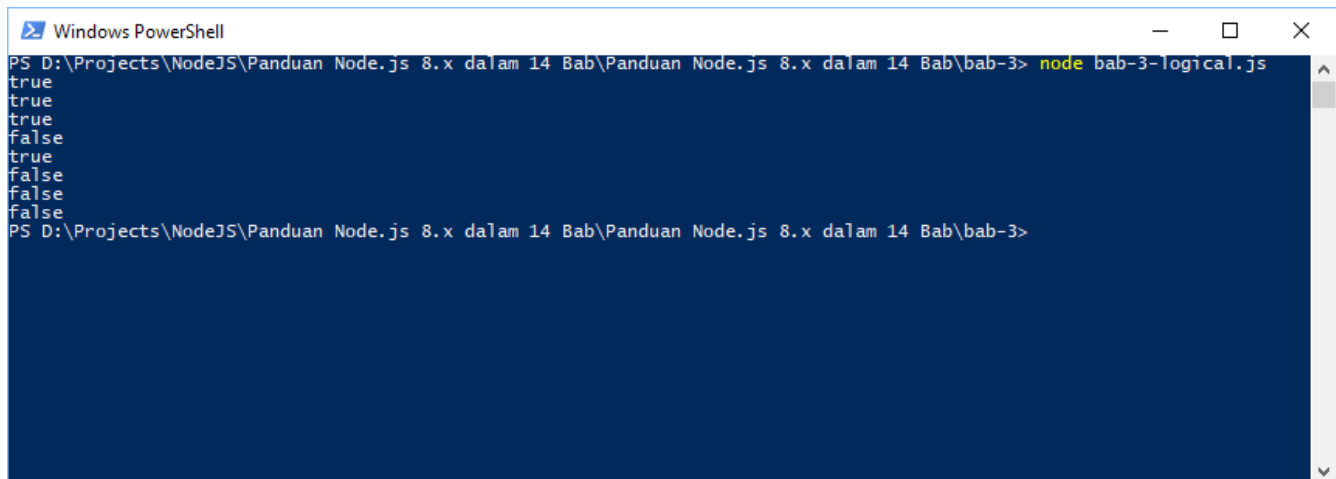
Untuk mencobanya, masuk ke folder bernama "bab-3", kemudian buat file baru di dalam folder tersebut dengan nama "bab-3-logical.js".

Isi file tadi dengan kode di atas.

Kemudian, buka PowerShell/Terminal pada folder bab-3 dan jalankan perintah ini:

```
node bab-3-logical .js
```

Hasilnya seperti ini:



```
Windows PowerShell
PS D:\Projects\NodeJS\Panduan Node.js 8.x dalam 14 Bab\Panduan Node.js 8.x dalam 14 Bab\bab-3> node bab-3-logical.js
true
true
true
false
true
false
false
false
false
PS D:\Projects\NodeJS\Panduan Node.js 8.x dalam 14 Bab\Panduan Node.js 8.x dalam 14 Bab\bab-3>
```

Loop

Javascript bisa melakukan pengulangan dengan while, do-while, dan for:

Kode di bawah ini mencetak angka mulai dari 10 sampai 1.

```
//loop dengan while
var a = 10;
while(a > 0){
  console.log(a);
  a--;
}

//loop dengan do while
```

```
a = 10;
do{
  console.log(a);
  a--;
} while (a > 0);

//loop dengan for
for(a = 10; a > 0; a--){
  console.log(a);
}
```

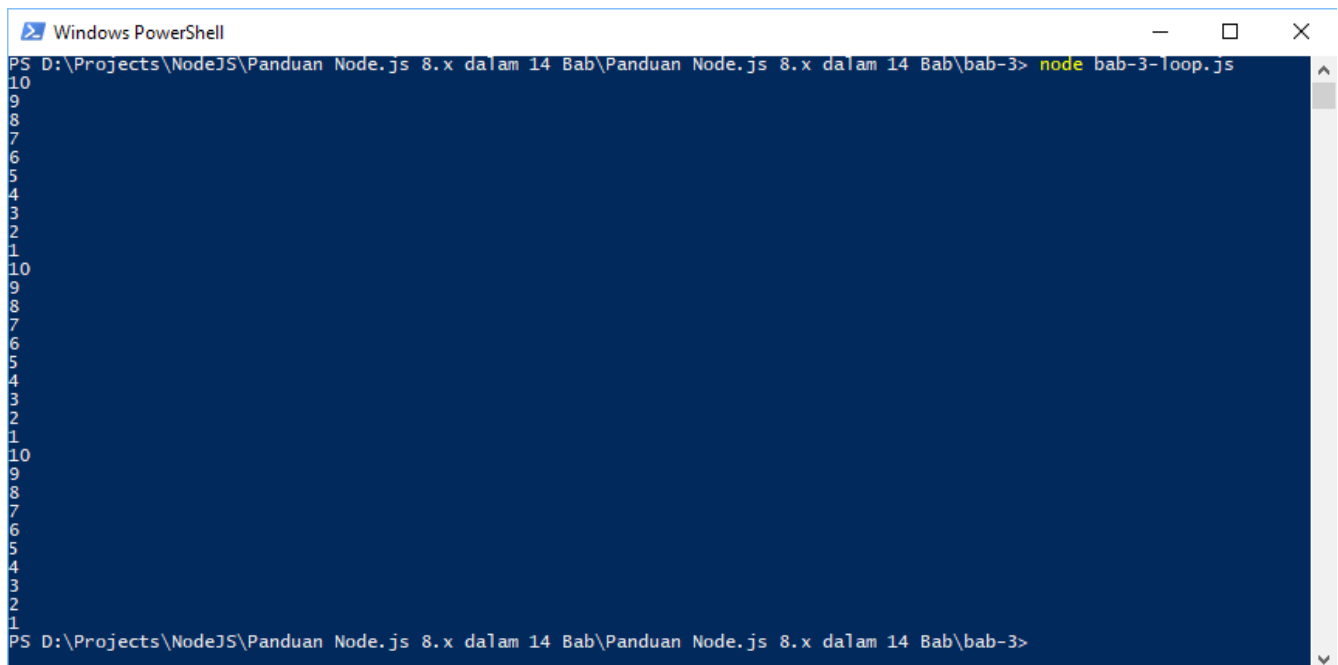
Untuk mencobanya, masuk ke folder bernama "bab-3", kemudian buat file baru di dalam folder tersebut dengan nama "bab-3-loop.js".

Isi file tadi dengan kode di atas.

Kemudian, buka PowerShell/Terminal pada folder bab-3 dan jalankan perintah ini:

```
node bab-3-loop .js
```

Hasilnya seperti ini:



```
Windows PowerShell
PS D:\Projects\NodeJS\Panduan Node.js 8.x dalam 14 Bab\Panduan Node.js 8.x dalam 14 Bab\bab-3> node bab-3-loop.js
10
9
8
7
6
5
4
3
2
1
10
9
8
7
6
5
4
3
2
1
10
9
8
7
6
5
4
3
2
1
PS D:\Projects\NodeJS\Panduan Node.js 8.x dalam 14 Bab\Panduan Node.js 8.x dalam 14 Bab\bab-3>
```

Ketiga jenis loop tadi hasilnya sama.

Fungsi

Javascript bisa menggunakan fungsi:

```
//mendeklarasikan fungsi
function doubleLogs(message){
  console.log(message);
  console.log(message);
}

//memanggil fungsi
doubleLogs("makan");

//fungsi dengan return value
function getCurrentFood(){
  return "sushi";
}

doubleLogs(getCurrentFood());

//fungsi bisa dijadikan variabel
var variabelFungsi = getCurrentFood;

console.log(variabelFungsi());
```

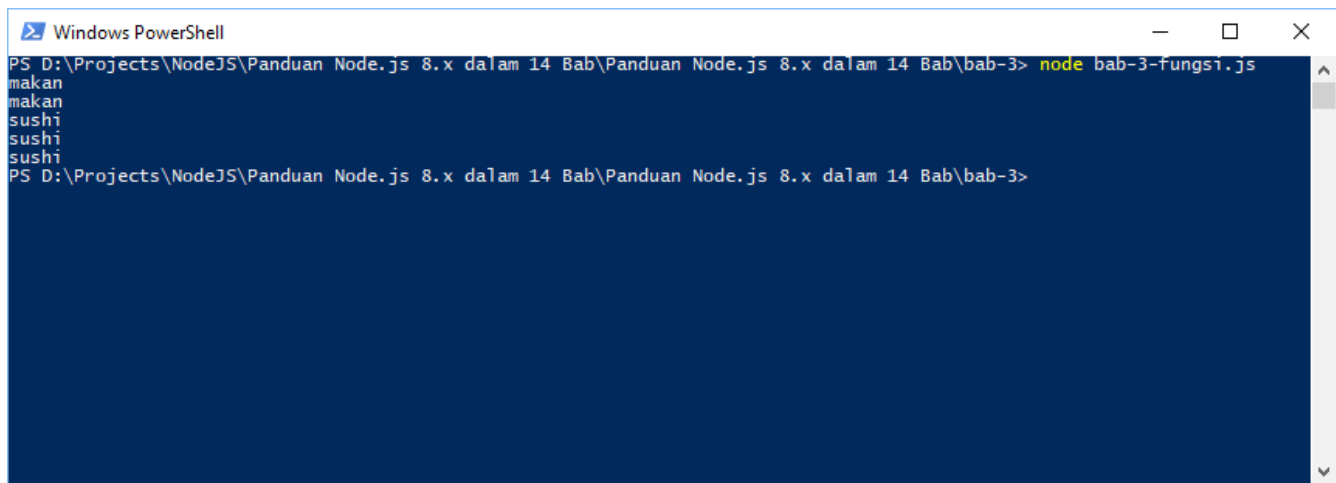
Untuk mencobanya, masuk ke folder bernama "bab-3", kemudian buat file baru di dalam folder tersebut dengan nama "bab-3-fungsi.js".

Isi file tadi dengan kode di atas.

Kemudian, buka PowerShell/Terminal pada folder bab-3 dan jalankan perintah ini:

```
node bab-3-fungsi .js
```

Hasilnya seperti ini:



```
Windows PowerShell
PS D:\Projects\NodeJS\Panduan Node.js 8.x dalam 14 Bab\Panduan Node.js 8.x dalam 14 Bab\bab-3> node bab-3-fungsi1.js
makan
sushi
sushi
sushi
PS D:\Projects\NodeJS\Panduan Node.js 8.x dalam 14 Bab\Panduan Node.js 8.x dalam 14 Bab\bab-3>
```

Class

Untuk membuat class di Javascript, saya lebih suka dengan cara lama.

Dengan cara ini kita bisa membuat member function dan variable.

Untuk membuat class, buat file baru di folder "bab-3" bernama "bab-3-modul-coffee-builder.js", kemudian isi dengan kode ini:

```
function CoffeeBuilder (coffeeName) {
    this.coffeeName = coffeeName;
}

CoffeeBuilder.prototype.getCoffeeName = function() {
    return this.coffeeName;
};

CoffeeBuilder.prototype.ambilGelas = function() {
    console.log("ambil gelas");
};

CoffeeBuilder.prototype.ambilKopi = function() {
    console.log("ambil kopi " + this.getCoffeeName());
};

CoffeeBuilder.prototype.masukkanKopiKeGelas = function() {
```

```
        console.log("masukkan kopi ke gelas");
    };

    CoffeeBuilder.prototype.seduhDenganAirPanas = function() {
        console.log("seduh kopi dengan air panas");
    };

    CoffeeBuilder.prototype.adukKopinya = function() {
        console.log("aduk");
    };

    CoffeeBuilder.prototype.jalankan = function() {
        this.ambilGelas();
        this.ambilKopi();
        this.masukkanKopiKeGelas();
        this.seduhDenganAirPanas();
        this.adukKopinya();
    };

    module.exports = CoffeeBuilder;
```

Untuk menggunakan class tadi, buat file baru di folder "bab-3" bernama "bab-3-run-coffee-builder.js", kemudian isi dengan kode ini:

```
var CoffeeBuilder = require("./bab-3-modul-coffee-builder.js");

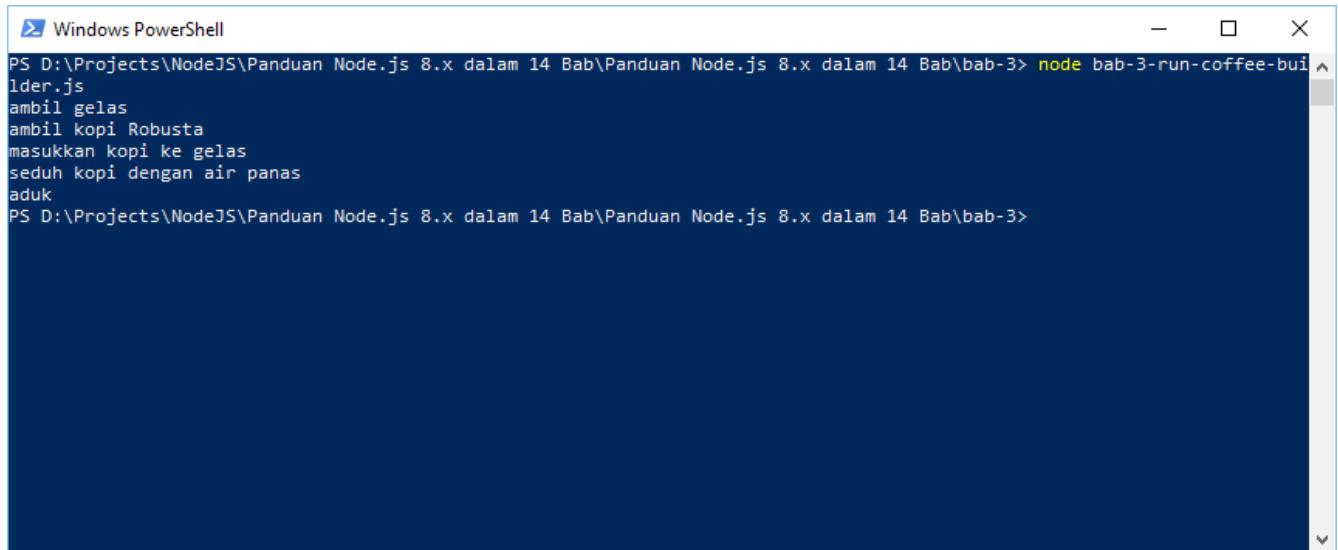
var cb = new CoffeeBuilder("Robusta");

cb.jalankan();
```

Jalankan kode tersebut dengan:

```
node bab-3-run-coffee-builder.js
```

Hasilnya seperti ini:

A screenshot of a Windows PowerShell terminal window. The title bar reads "Windows PowerShell". The command prompt shows the current directory as "PS D:\Projects\NodeJS\Panduan Node.js 8.x dalam 14 Bab\Panduan Node.js 8.x dalam 14 Bab\bab-3". The command executed is "node bab-3-run-coffee-builder.js". The output of the script is displayed on the following lines: "ambil gelas", "ambil kopi Robusta", "masukkan kopi ke gelas", "seduh kopi dengan air panas", and "aduk". The prompt then returns to "PS D:\Projects\NodeJS\Panduan Node.js 8.x dalam 14 Bab\Panduan Node.js 8.x dalam 14 Bab\bab-3>".

```
Windows PowerShell
PS D:\Projects\NodeJS\Panduan Node.js 8.x dalam 14 Bab\Panduan Node.js 8.x dalam 14 Bab\bab-3> node bab-3-run-coffee-builder.js
ambil gelas
ambil kopi Robusta
masukkan kopi ke gelas
seduh kopi dengan air panas
aduk
PS D:\Projects\NodeJS\Panduan Node.js 8.x dalam 14 Bab\Panduan Node.js 8.x dalam 14 Bab\bab-3>
```

Promise

Karena banyak fungsi asynchronous di Node.js, maka terkadang kita agak kesulitan menulis kode yang panjang dan mendalam.

Akibatnya, callback menjadi semakin bersarang.

Solusinya, digunakan promise.

Dengan promise, fungsi yang dipanggil dengan cara semacam ini:

```
listUser(numOfUser, function(theList, error){
  if(error){
    console.log(error);
  }
  console.log(theList);
});
```

Bisa menjadi begini:

```
listUser(numOfUser).then(function(theList){
  console.log(theList);
}).catch(function(error){
  console.log(error);
});
```


Untuk membuat fungsi promise dari fungsi yang tidak promise begini caranya:

```
var myPromise = function(){  
  return new Promise(function(resolve, reject){  
    setTimeout(function(){ //BISA DIGANTI DENGAN FUNGSI APAPUN  
      resolve('Promise di-resolve'); //COBA DEAKTIFKAN INI dengan menambah '//'  
      //reject('Promise di-reject'); //COBA AKTIFKAN INI dengan menghapus '//'  
    }, 2500);  
  });  
}
```

Fungsi tersebut membungkus fungsi setTimeout yang tidak promise.

Jika kondisinya benar, gunakan resolve, sedangkan jika salah, gunakan reject.

Penggunaannya seperti ini:

```
myPromise().then((message) => {  
  console.log('Success: ', message);  
}).catch((error) => {  
  console.log('Error: ', error);  
});
```

Sekarang coba jalankan kode tersebut, maka hasilnya:

```
Success:  Promise di-resolve
```

```
Windows PowerShell
PS D:\Projects\NodeJS\Panduan Node.js 8.x dalam 14 Bab\Panduan Node.js 8.x dalam 14 Bab\bab-3> node bab-3-promise.js
Success: Promise di-resolve
PS D:\Projects\NodeJS\Panduan Node.js 8.x dalam 14 Bab\Panduan Node.js 8.x dalam 14 Bab\bab-3>
```

Jika resolve dihapus dan diganti jadi reject:

```
var myPromise = function(){
  return new Promise(function(resolve, reject){
    setTimeout(function(){ //BISA DIGANTI DENGAN FUNGSI APAPUN
      reject('Promise di-reject'); //COBA AKTIFKAN INI dengan menghapus '//'
    }, 2500);
  });
}
```

Maka hasilnya seperti ini:

```
Error: Promise di-reject
```

Bisa kita lihat bahwa argument dari resolve akan dimasukkan ke argument dari then.

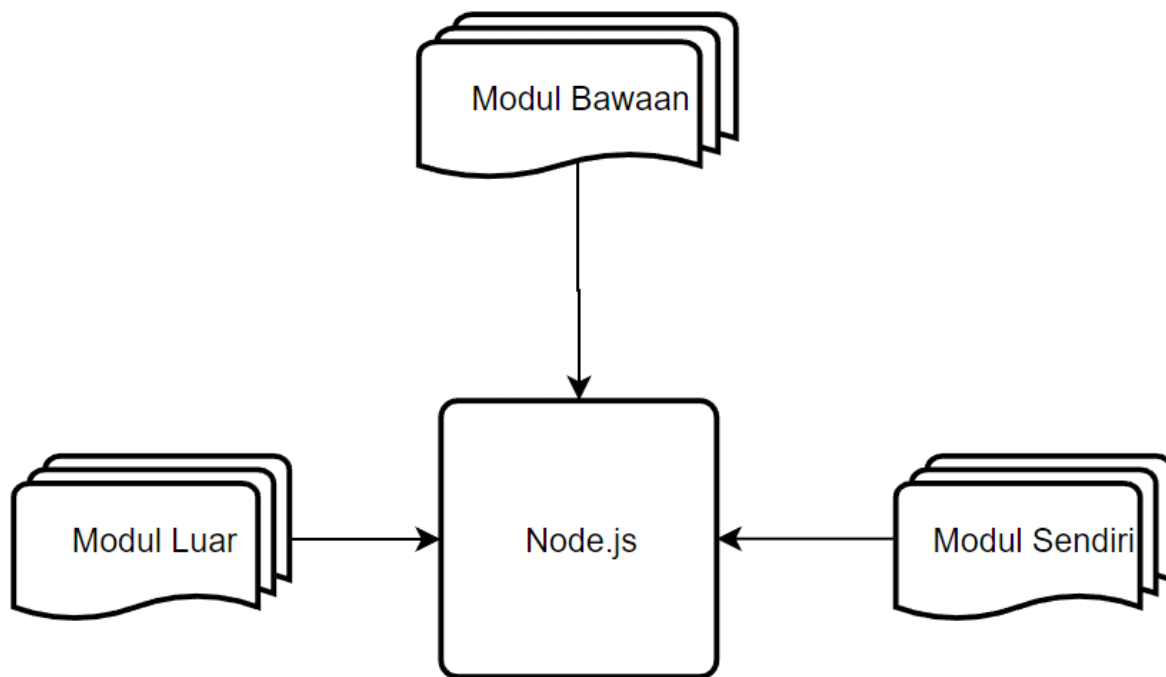
Sementara itu, argument dari reject akan dimasukkan ke argument dari catch.

Bab 4. Modul-Modul Node.js

Pada dasarnya, modul Node.js adalah library Javascript.

Ada modul yang sudah ada secara default bersama Node.js yang telah ter-install.

Ada juga modul yang harus di-install dahulu agar dapat digunakan.



Node.js dan Modul-Modulnya

Modul yang pertama tadi tidak perlu di-install dengan NPM agar dapat digunakan, sedangkan modul yang terakhir harus di-install terlebih dahulu dengan NPM.

Modul bawaan maupun yang sudah ter-install kemudian di-include ke dalam script Node.js kita dengan fungsi `require()`.

Akan tetapi ada juga modul bawaan yang tidak perlu `require()`, misalnya `Buffer`.

Selain itu, kita juga bisa membuat modul sendiri dan meng-include-nya pada script lain.

Beberapa Modul Bawaan

File System

Modul ini adalah modul yang digunakan untuk memanipulasi file.

Untuk meng-include-nya:

```
var fs = require("fs");
```

Di situ, kita meng-include modul "fs" dengan require.

Dengan meng-include-nya, kita bisa menggunakan modul "fs" di dalam script peng-include-nya.

Semua method dari modul File System ada yang synchronous dan asynchronous.

Yang asynchronous tidak akan mem-block program ketika dieksekusi, sedangkan yang synchronous akan mem-block-nya.

Untuk mencobanya, buat file baru bernama "bab-4-teks-untuk-dibaca-asynchronous.txt", kemudian isi dengan ini:

```
file ini hanya untuk contoh.  
silakan dibaca secara ASYNCHRONOUS.
```

Kemudian buat file baru bernama "bab-4-teks-untuk-dibaca-synchronous.txt", kemudian isi dengan ini:

```
file ini hanya untuk contoh.  
silakan dibaca secara SYNCHRONOUS.
```

Kemudian, buat file baru bernama "bab-4-fs-sync-async.js", kemudian isi dengan kode ini:

```
var fs = require("fs");  
  
console.log("START");  
  
//Membaca secara asynchronous  
fs.readFile('bab-4-teks-untuk-dibaca-asynchronous.txt', function (err, data) {  
    if (err) {  
        return console.error(err);  
    }  
})
```

```

    }
    console.log("File ini dibaca secara asynchronous: " + data.toString());
});

//Membaca secara synchronous
var data = fs.readFileSync('bab-4-teks-untuk-dibaca-synchronous.txt');
console.log("File ini dibaca secara synchronous: " + data.toString());

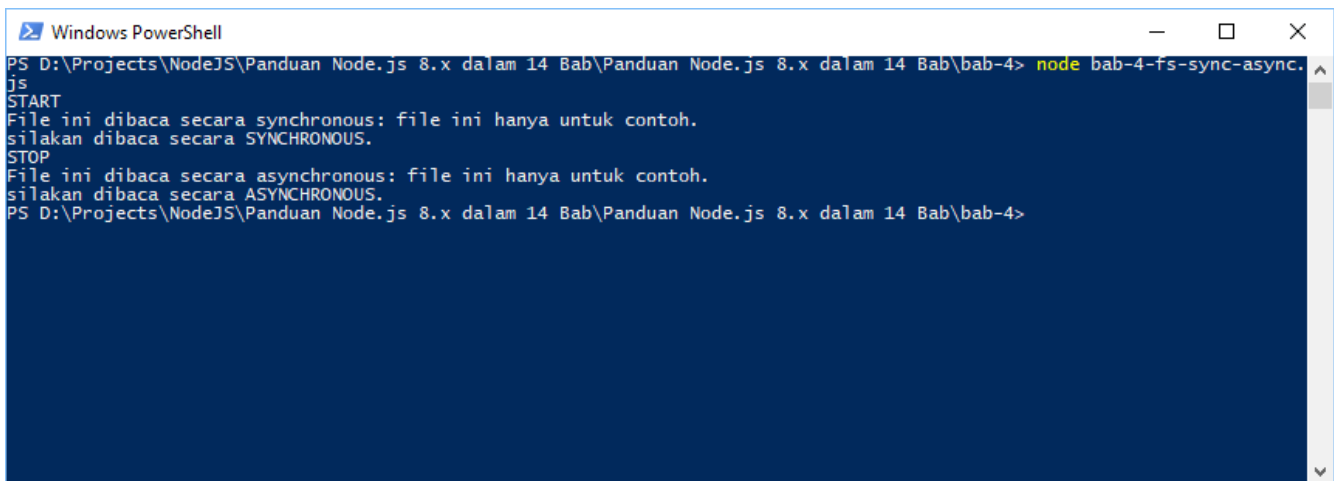
console.log("STOP");

```

Jalankan script ini dengan cara:

```
node bab-4-fs-sync-async.js
```

Nanti output-nya seperti ini:



```

Windows PowerShell
PS D:\Projects\NodeJS\Panduan Node.js 8.x dalam 14 Bab\Panduan Node.js 8.x dalam 14 Bab\bab-4> node bab-4-fs-sync-async.js
START
File ini dibaca secara synchronous: file ini hanya untuk contoh.
silakan dibaca secara SYNCHRONOUS.
STOP
File ini dibaca secara asynchronous: file ini hanya untuk contoh.
silakan dibaca secara ASYNCHRONOUS.
PS D:\Projects\NodeJS\Panduan Node.js 8.x dalam 14 Bab\Panduan Node.js 8.x dalam 14 Bab\bab-4>

```

Tampak bahwa program akan meneruskan instruksi ke bagian yang synchronous dulu sambil menunggu yang asynchronous selesai dibaca.

Dengan demikian, yang synchronous akan tampil terlebih dahulu.

Namun, apabila kita meletakkan loop di bawah kode ini:

```

//Membaca secara synchronous
var data = fs.readFileSync('bab-4-teks-untuk-dibaca-synchronous.txt');
console.log("File ini dibaca secara synchronous: " + data.toString());
console.log("STOP");

```

```
while(1); //infinite loop! COBA TAMBAHKAN INI
```

Akan berakibat fungsi asynchronous read dihambat dan dalam kasus ini tidak akan dipanggil karena infinite loop.

Hal itu disebabkan call stack harus kosong terlebih dahulu sebelum callback dipanggil.

Dan itu juga berlaku bagi fungsi lain yang asynchronous.

HTTP

Modul ini adalah modul yang digunakan untuk membuat server atau client HTTP.

Untuk membuat HTTP server, buat file "bab-4-http-server.js", kemudian isi dengan:

```
var http1 = require('http');

let pesan = 'Ini adalah server http!';
http1.createServer(function (req, res) {
    res.write(pesan); //tuliskan
    res.end(); //akhiri
}).listen(8080); //gunakan port 8080
```

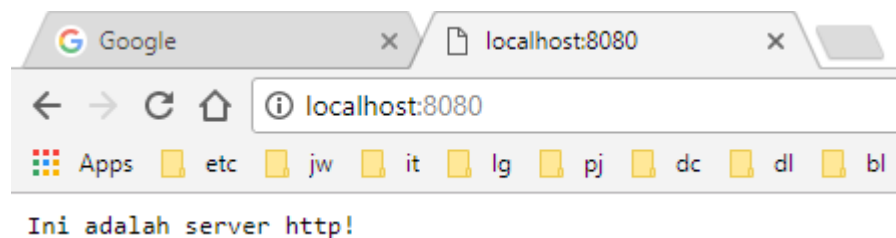
Kemudian, jalankan perintah ini di PowerShell/Terminal:

```
node bab-4-http-server.js
```

Setelah dijalankan, buka browser kita ke:

```
http://localhost:8080
```

Nanti output-nya:



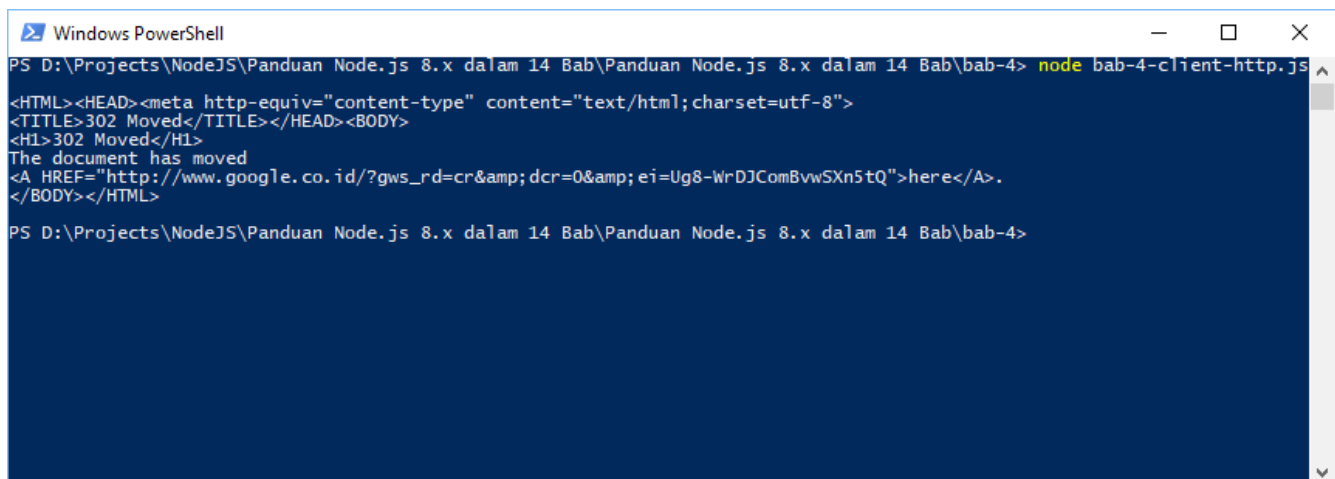
Untuk membuat HTTP Client, buat file "bab-4-client-http.js", kemudian isi dengan:

```
var http1 = require('http');
var req = http1.request({
  host: 'www.google.com',
  port: '80',
  path: '/'
}, function(response){
  var isiHTML = '';
  response.on('data', function(potonganData) {
    //Akumulasikan data
    isiHTML += potonganData;
  });
  response.on('end', function() {
    //Data telah diterima secara komplit
    console.log(isiHTML);
  });
});
req.end();
```

Selanjutnya, buka PowerShell/Terminal dan jalankan perintah:

```
node bab-4-client-http.js
```

Hasilnya seperti ini:



```
Windows PowerShell
PS D:\Projects\NodeJS\Panduan Node.js 8.x dalam 14 Bab\Panduan Node.js 8.x dalam 14 Bab\bab-4> node bab-4-client-http.js
<HTML><HEAD><meta http-equiv="content-type" content="text/html; charset=utf-8">
<TITLE>302 Moved</TITLE></HEAD><BODY>
<H1>302 Moved</H1>
The document has moved
<A HREF="http://www.google.co.id/?gws_rd=cr&dc=0&ei=Ug8-WrDJComBvwSXn5tQ">here</A>.
</BODY></HTML>
PS D:\Projects\NodeJS\Panduan Node.js 8.x dalam 14 Bab\Panduan Node.js 8.x dalam 14 Bab\bab-4>
```

Events

Di dalam pemrograman, seringkali kita menghadapi berbagai event.

Misalnya saat mau membaca file, ada event "file dibuka".

Sementara itu, saat pembacaan file terjadi, ada event "data sedang dibaca" dan seterusnya hingga ada event "file ditutup".

Node.js memfasilitasi kita dengan modul Events sehingga kita bisa menembakkan dan menerima event dari fungsi yang kita buat sendiri.

Sebagai contoh, kita ingin menembakkan event pada saat 3 detik setelah sebuah script dijalankan:

```
setTimeout(function(){  
  console.log("sudah 3 detik");  
  //tembakkan event di sini  
}, 3000);
```

Maka pada bagian yang ditandai "//" kita hanya perlu menambahkan fungsi emit dari modul EventEmitter.

Sekarang kita akan mencoba modul EventEmitter.

Buatlah sebuah file baru di dalam folder "bab-4" bernama "bab-4-events.js".

Isinya seperti ini:

```
console.log("START");  
  
var EventEmitter = require("events");  
var eventEmitter = new EventEmitter();  
  
//buat event handler  
var openHandler = function(){  
  console.log("file opened");  
};  
  
var readHandler = function(){  
  console.log("reading...");  
};  
  
var closeHandler = function(){  
  console.log("file closed");  
};
```



```
//pasang event handler ke events
eventEmitter.on('open', openHandler);
eventEmitter.on('read', readHandler);
eventEmitter.on('close', closeHandler);

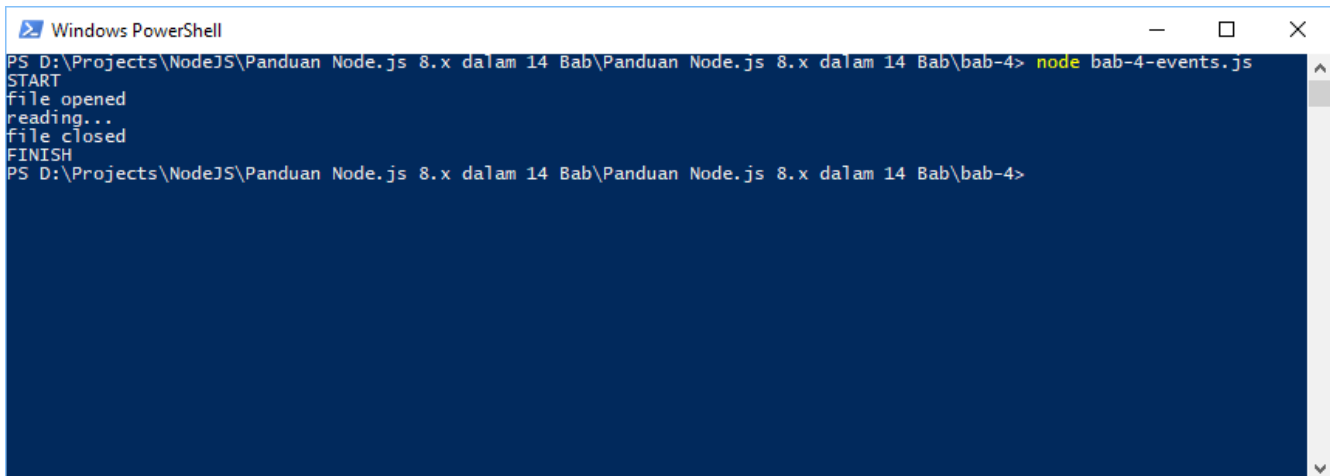
//tembakkan event
eventEmitter.emit('open');
eventEmitter.emit('read');
eventEmitter.emit('close');

console.log("FINISH");
```

Kemudian jalankan script tersebut:

```
node bab-4-events.js
```

Output-nya seperti ini:

A screenshot of a Windows PowerShell terminal window. The title bar reads 'Windows PowerShell'. The command prompt shows the directory 'PS D:\Projects\NodeJS\Panduan Node.js 8.x dalam 14 Bab\Panduan Node.js 8.x dalam 14 Bab\bab-4' and the command 'node bab-4-events.js'. The output of the script is displayed as follows:

```
START
file opened
reading...
file closed
FINISH
PS D:\Projects\NodeJS\Panduan Node.js 8.x dalam 14 Bab\Panduan Node.js 8.x dalam 14 Bab\bab-4>
```

Handler-handler tadi:

```
//buat event handler
var openHandler = function(){
  console.log("file opened");
};

var readHandler = function(){
  console.log("reading...");
};
```

```
var closeHandler = function(){  
  console.log("file closed");  
};
```

Dipanggil di fungsi emit:

```
//tembakkan event  
eventEmitter.emit('open');  
eventEmitter.emit('read');  
eventEmitter.emit('close');
```

Modul-Modul Bawaan Lainnya

Masih banyak modul-modul lain yang disertakan Node.js secara default.

Kita bisa mempelajarinya dari dokumentasi Node.js di:

<https://nodejs.org/dist/latest-v8.x/docs/api/>

Modul Luar

Selain modul bawaan, ada juga modul luar yang harus di-install terlebih dahulu sebelum dapat digunakan.

Untuk meng-install-nya, kita menggunakan perintah NPM:

```
npm install <nama_package>
```

Jadi, jika kita ingin meng-install Express.js:

```
npm install express
```

NPM bisa menginstall modul secara local maupun global.

Secara default, NPM menginstall modul secara local.

Secara local, modul akan di-install dalam folder "node_modules" yang ada di dalam folder dimana perintah NPM dijalankan.

Jadi, jika kita menjalankan perintah NPM dalam folder "C:\Nodejs", maka lokasi folder "node_modules" berada di dalam folder tersebut.

Agar modul terinstall secara global, tambahkan parameter -g:

```
npm install express -g
```

Dengan cara ini, modul terinstall di dalam folder global yang berbeda-beda lokasinya di setiap OS.

Jika kita menggunakan Windows, maka folder global tersebut ada di "C:\Users\<nama_user>\AppData\Roaming\npm\node_modules"

Jika kita menggunakan Linux, maka folder itu ada di "/usr/local/lib/node" atau di "/usr/local/lib/node_modules"

Lokasi folder global tersebut bisa dicek dengan perintah ini:

```
npm root -g
```

Penginstalan modul juga dapat diatur sehingga modul yang terinstall didaftarkan dalam bagian "dependencies" pada file "package.json".

Jika sudah didaftarkan di sana, kita bisa menginstall kembali semua modul dengan satu perintah:

```
npm install
```

Agar modul yang di-install didaftarkan dalam package.json, gunakan parameter --save:

```
npm install express --save
```

Modul juga dapat di-uninstall dengan cara ini:

```
npm uninstall <nama_package>
```

Jadi jika ingin meng-uninstall Express.js, begini caranya:

```
npm uninstall express
```

Modul Sendiri

Kita juga dapat membuat modul sendiri.

Misalnya, kita ingin membuat modul yang mengembalikan sebuah pesan teks:

```
exports.pesan = function () {  
  return "ini adalah pesan saya";  
};
```

Simpan kode tersebut dengan nama "bab-4-modulpesan.js".

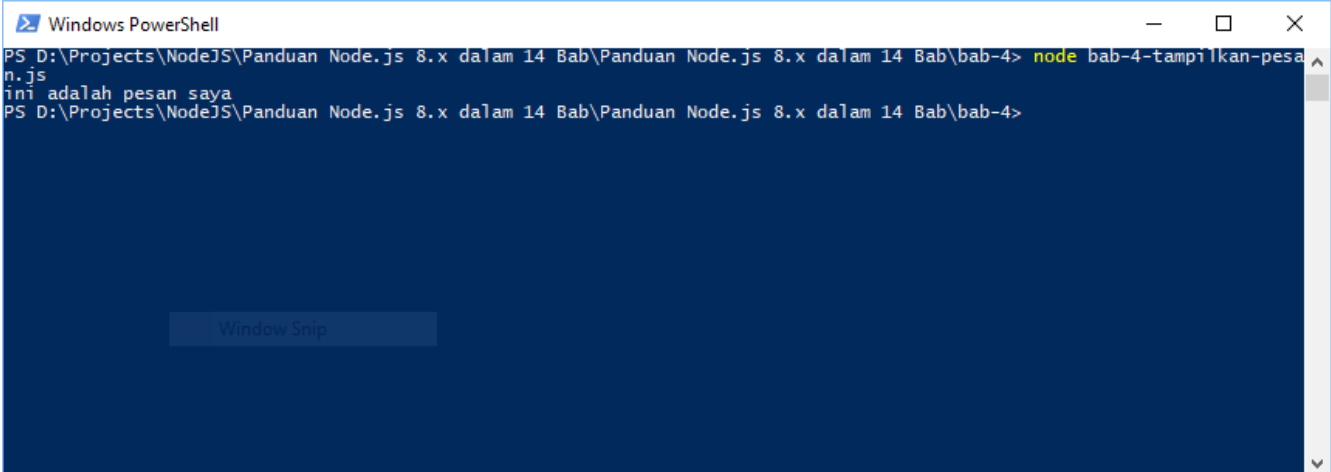
Untuk meng-include-nya, gunakan require pada file lain bernama "bab-4-tampilkan-pesan.js":

```
var modulpesan = require('./bab-4-modulpesan');  
  
console.log(modulpesan.pesan());
```

Kemudian, jalankan perintah ini di PowerShell/Terminal:

```
node bab-4-tampilkan-pesan.js
```

Kode itu akan menampilkan:



The screenshot shows a Windows PowerShell window with a dark blue background. The title bar reads "Windows PowerShell". The command prompt shows the current directory as "PS D:\Projects\NodeJS\Panduan Node.js 8.x dalam 14 Bab\Panduan Node.js 8.x dalam 14 Bab\bab-4\". The user has entered the command "node bab-4-tampilkan-pesan.js". The output of the command is "ini adalah pesan saya". A "Window Snip" watermark is visible in the center of the terminal window.

Bab 5. Node.js Globals

Node.js memiliki globals yang dapat diakses tanpa menggunakan require.

Bahkan require sendiri merupakan globals dalam bentuk fungsi.

Di sini akan dibahas beberapa globals milik Node.js.

__dirname

Ini adalah nama direktori dari script yang sedang dijalankan.

Nilainya sama dengan `path.dirname(__filename)`.

Contohnya, ketika kita menjalankan script ini di file "bab-5-dirname.js" dalam folder "D:/Node":

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

// Menampilkan "D:/Node"
console.log(__dirname);

// Menampilkan "D:/Node"
console.log(path.dirname(__filename));
```

Hasilnya bisa berbeda tergantung di mana kita menjalankan script tersebut.

__filename

Ini adalah nama file path (direktori + nama file) dari script yang dijalankan.

Contohnya, ketika kita menjalankan script ini di file "bab-5-filename.js" dalam folder "D:/Node":

```
console.log(__filename);
```

Maka itu akan menampilkan "D:/Node/bab-4-filename.js"

Tapi tidak berarti bahwa `__filename` hanya menampilkan file path dari script yang dijadikan input dalam perintah:

```
node bab-4-filename.js
```

Seandainya script "bab-5-filename.js" memiliki dependency pada script lain dengan lokasi di folder yang lain, maka `__filename` di script lain tersebut bernilai file path dari script lain tersebut.

Jadi, script "D:/Node/bab-4-deps.js" yang diimport oleh "D:/Node/bab-5-filename.js" nilai `__filename` nya adalah "D:/Node/bab-4-deps.js".

console

Ini digunakan untuk melakukan debugging maupun program Node.js berbasis command line interface.

Tugasnya adalah menampilkan pesan-pesan informatif.

Contoh penggunaannya ada di file "bab-4-console.js":

```
// Menampilkan "hello world" ke stdout
console.log('hello world');

// Menampilkan "hello world" ke stdout
console.log('hello %s', 'world');

// Menampilkan "[Error: Error, kurang banyak]" ke stderr
console.error(new Error('Error, kurang banyak'));

// Menampilkan "Anonim" ke stderr
const name = 'Anonim';
console.warn(`Go ${name}! Go!`);
```

module

Ini digunakan untuk mendapatkan referensi ke current module.

exports

Ini adalah penyederhanaan dari `module.exports`.

Kita telah membahas ini pada Bab 4, di bagian "Modul Sendiri".

global (tanpa "s")

Ini digunakan untuk membuat variabel global.

Pada browser Javascript, scope terluar merupakan variabel global.

Tapi tidak demikian di Node.js. Scope terluar adalah local bagi modul tersebut.

Untuk membuatnya menjadi global, digunakan:

```
global.namavariabel = "sebuah string global";
```

process

Ini digunakan untuk mendapatkan informasi dan mengendalikan process Node.js yang sedang dijalankan.

Process digunakan salah satunya untuk menangkap event dari Node.js.

Selain itu, process juga bisa digunakan untuk memfilter argument dari command line, mendapatkan pid, menghentikan process lain, dan lain-lain.

Contohnya bisa dilihat pada file "bab-5-process.js" pada source code yang disertakan:

```
setTimeout(function(){
  console.log("sudah 3 detik");
}, 3000);

process.on('exit', (code) => {
  console.log(`Arsitektur processor ini adalah ${process.arch}`);
  console.log(`Keluar dengan exit code: ${code}`);
});
```

require

Ini adalah fungsi yang digunakan untuk mengimpor modul.

Kita telah mencobanya di bab-bab sebelum ini.

setTimeout

Ini adalah fungsi yang digunakan untuk menjadwalkan callbacknya.

Contohnya bisa dilihat pada file "bab-5-settimeout.js" pada source code yang disertakan:

```
setTimeout(function(){
  console.log("3 detik terlewati");
}, 3000);
```

setInterval

Ini adalah fungsi yang digunakan untuk menjadwalkan callbacknya secara berulang-ulang.

Contohnya bisa dilihat pada file "bab-5-setinterval.js" pada source code yang disertakan:

```
setInterval(function(){
  console.log("sudah 3 detik, ulang lagi ya...");
}, 3000);
```

setImmediate

Ini akan menjalankan callback sesegera mungkin.

Contohnya bisa dilihat pada file "bab-5-setimmediate.js" pada source code yang disertakan:

```
var setTimeoutCallback = function(){
  console.log("setTimeoutCallback dipanggil");
};

var callbackBiasa = function(){
  console.log("callbackBiasa dipanggil");
};

var setImmediateCallback = function(){
  console.log("setImmediateCallback dipanggil");
};
```



```
setTimeout(setTimeoutCallback, 5000);

setImmediate(setImmediateCallback);

callbackBiasa();
```

Output-nya:

```
callbackBiasa dipanggil
setImmediateCallback dipanggil
setTimeoutCallback dipanggil
```

Tampak bahwa setImmediateCallback dipanggil setelah callbackBiasa. walaupun urutannya lebih dahulu setImmediate.

Globals Lainnya

Selain global-global tadi, masih ada yang lain, tapi tidak dibahas di sini.

Untuk mengetahui semua global yang ada, caranya dengan masuk ke REPL dengan perintah:

```
node
```

Kemudian ketik:

```
global
```

Kemudian tekan enter.

Nanti akan tampil output semua global yang ada.

Output-nya seperti ini:

```
{ console: [Getter],
  DTRACE_NET_SERVER_CONNECTION: [Function],
  DTRACE_NET_STREAM_END: [Function],
  DTRACE_HTTP_SERVER_REQUEST: [Function],
  DTRACE_HTTP_SERVER_RESPONSE: [Function],
  DTRACE_HTTP_CLIENT_REQUEST: [Function],
  DTRACE_HTTP_CLIENT_RESPONSE: [Function],
  COUNTER_NET_SERVER_CONNECTION: [Function],
  COUNTER_NET_SERVER_CONNECTION_CLOSE: [Function],
  COUNTER_HTTP_SERVER_REQUEST: [Function],
  COUNTER_HTTP_SERVER_RESPONSE: [Function],
```

```

COUNTER_HTTP_CLIENT_REQUEST: [Function],
COUNTER_HTTP_CLIENT_RESPONSE: [Function],
global: [Circular],
process:
  process {
    title: 'Administrator: Windows PowerShell',
    version: 'v8.9.2',
    moduleLoadList:
      [ 'Binding contextify',
        'Binding natives',
        'Binding config',
        'NativeModule events',
        'Binding async_wrap',
        'Binding icu',
        'NativeModule util',
        'NativeModule internal/errors',
        'NativeModule internal/encoding',
        'NativeModule internal/util',
        'Binding util',
        'Binding constants',
        'NativeModule internal/util/types',
        'Binding buffer',
        'NativeModule buffer',
        'NativeModule internal/buffer',
        'Binding uv',
        'NativeModule internal/process',
        'NativeModule internal/process/warning',
        'NativeModule internal/process/next_tick',
        'NativeModule async_hooks',
        'NativeModule internal/process/promises',
        'NativeModule internal/process/stdio',
        'Binding performance',
        'NativeModule perf_hooks',
        'NativeModule internal/linkedlist',
        'NativeModule internal/inspector_async_hook',
        'Binding inspector',
        'NativeModule timers',
        'Binding timer_wrap',
        'NativeModule assert',
        'NativeModule module',
        'NativeModule internal/module',
        'NativeModule internal/url',
        'NativeModule internal/querystring',
        'NativeModule querystring',
        'Binding url',
        'NativeModule vm',
        'NativeModule fs',
        'NativeModule path',
        'Binding fs',
        'NativeModule stream',
        'NativeModule internal/streams/legacy',
        'NativeModule _stream_readable',

```

```

'NativeModule internal/streams/BufferList',
'NativeModule internal/streams/destroy',
'NativeModule _stream_writable',
'NativeModule _stream_duplex',
'NativeModule _stream_transform',
'NativeModule _stream_passthrough',
'Binding fs_event_wrap',
'NativeModule internal/fs',
'NativeModule internal/loader/Loader',
'NativeModule internal/loader/ModuleWrap',
'Internal Binding module_wrap',
'NativeModule internal/loader/ModuleMap',
'NativeModule internal/loader/ModuleJob',
'NativeModule internal/safe_globals',
'NativeModule internal/loader/ModuleRequest',
'NativeModule url',
'NativeModule internal/loader/search',
'NativeModule console',
'Binding tty_wrap',
'NativeModule tty',
'NativeModule net',
'NativeModule internal/net',
'Binding cares_wrap',
'Binding tcp_wrap',
'Binding pipe_wrap',
'Binding stream_wrap',
'NativeModule dns',
'NativeModule readline',
'NativeModule string_decoder',
'NativeModule internal/readline',
'Binding signal_wrap',
'NativeModule internal/repl',
'NativeModule repl',
'NativeModule domain',
'NativeModule os',
'NativeModule internal/os',
'Binding os' ],
  versions: { http_parser: '2.7.0',
node: '8.9.2',
v8: '6.1.534.48',
uv: '1.15.0',
zlib: '1.2.11',
ares: '1.10.1-DEV',
modules: '57',
nghttp2: '1.25.0',
openssl: '1.0.2m',
icu: '59.1',
unicode: '9.0',
cldr: '31.0.1',
tz: '2017b' },
  arch: 'x64',
  platform: 'win32',

```

```

release:
{ name: 'node',
  lts: 'Carbon',
  sourceUrl: 'https://nodejs.org/download/release/v8.9.2/node-v8.9.2.tar.gz',
  headersUrl: 'https://nodejs.org/download/release/v8.9.2/node-v8.9.2-
headers.tar.gz',
  libUrl: 'https://nodejs.org/download/release/v8.9.2/win-x64/node.lib' },
argv: [ 'D:\\SDK\\NodeJS\\node.exe' ],
execArgv: [],
env:
{ ALLUSERSPROFILE: 'C:\\ProgramData',
  APPDATA: 'C:\\Users\\laptop.baru.bagus\\AppData\\Roaming',
  CommonProgramFiles: 'C:\\Program Files\\Common Files',
  'CommonProgramFiles(x86)': 'C:\\Program Files (x86)\\Common Files',
  CommonProgramW6432: 'C:\\Program Files\\Common Files',
  COMPUTERNAME: 'DESKTOP-IG2OVAK',
  ComSpec: 'C:\\WINDOWS\\system32\\cmd.exe',
  GTK_BASEPATH: 'C:\\Program Files (x86)\\GtkSharp\\2.12\\',
  HOMEDRIVE: 'C:',
  HOMEPATH: '\\Users\\laptop.baru.bagus',
  JAVA_HOME: 'D:\\SDK\\Java\\jre-9.0.1\\bin',
  KMP_DUPLICATE_LIB_OK: 'TRUE',
  LOCALAPPDATA: 'C:\\Users\\laptop.baru.bagus\\AppData\\Local',
  LOGONSERVER: '\\\\DESKTOP-IG2OVAK',
  MKL_SERIAL: 'YES',
  NUMBER_OF_PROCESSORS: '8',
  OneDrive: 'C:\\Users\\laptop.baru.bagus\\OneDrive',
  OS: 'Windows_NT',
  Path: 'C:\\ProgramData\\Oracle\\Java\\javapath;D:\\SDK\\Python27\\;D:\\SDK\\
Python27\\Scripts;C:\\WINDOWS\\system32;C:\\WINDOWS;C:\\WINDOWS\\System32\\Wbem;C:\\
WINDOWS\\System32\\WindowsPowerShell\\v1.0\\;C:\\Program Files (x86)\\GtkSharp\\2.12\\
bin;D:\\SDK\\NodeJS\\;C:\\Program Files\\PuTTY\\;C:\\Program Files (x86)\\Wolfram
Research\\WolframScript\\;C:\\Users\\laptop.baru.bagus\\AppData\\Local\\Microsoft\\
WindowsApps;C:\\Program Files\\Microsoft VS Code\\bin;C:\\Users\\laptop.baru.bagus\\
AppData\\Roaming\\npm',
  PATHEXT: '.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC;.CPL',
  PROCESSOR_ARCHITECTURE: 'AMD64',
  PROCESSOR_IDENTIFIER: 'Intel64 Family 6 Model 60 Stepping 3, GenuineIntel',
  PROCESSOR_LEVEL: '6',
  PROCESSOR_REVISION: '3c03',
  ProgramData: 'C:\\ProgramData',
  ProgramFiles: 'C:\\Program Files',
  'ProgramFiles(x86)': 'C:\\Program Files (x86)',
  ProgramW6432: 'C:\\Program Files',
  PSModulePath: 'C:\\Users\\laptop.baru.bagus\\Documents\\WindowsPowerShell\\
Modules;C:\\Program Files\\WindowsPowerShell\\Modules;C:\\WINDOWS\\system32\\
WindowsPowerShell\\v1.0\\Modules',
  PUBLIC: 'C:\\Users\\Public',
  SystemDrive: 'C:',
  SystemRoot: 'C:\\WINDOWS',
  TEMP: 'C:\\Users\\LAPTOP~1.BAG\\AppData\\Local\\Temp',
  TMP: 'C:\\Users\\LAPTOP~1.BAG\\AppData\\Local\\Temp',

```

```

    USERDOMAIN: 'DESKTOP-IG20VAK',
    USERDOMAIN_ROAMINGPROFILE: 'DESKTOP-IG20VAK',
    USERNAME: 'laptop.baru.bagus',
    USERPROFILE: 'C:\\Users\\laptop.baru.bagus',
    VBOX_MSI_INSTALL_PATH: 'C:\\Program Files\\Oracle\\VirtualBox\\',
    windir: 'C:\\WINDOWS' },
pid: 1652,
features:
  { debug: false,
    uv: true,
    ipv6: true,
    tls_npn: true,
    tls_alpn: true,
    tls_sni: true,
    tls_ocsp: true,
    tls: true },
_needImmediateCallback: false,
execPath: 'D:\\SDK\\NodeJS\\node.exe',
debugPort: 9229,
_startProfilerIdleNotifier: [Function: _startProfilerIdleNotifier],
_stopProfilerIdleNotifier: [Function: _stopProfilerIdleNotifier],
_getActiveRequests: [Function: _getActiveRequests],
_getActiveHandles: [Function: _getActiveHandles],
reallyExit: [Function: reallyExit],
abort: [Function: abort],
chdir: [Function: chdir],
cwd: [Function: cwd],
umask: [Function: umask],
_kill: [Function: _kill],
_debugProcess: [Function: _debugProcess],
_debugPause: [Function: _debugPause],
_debugEnd: [Function: _debugEnd],
hrtime: [Function: hrtime],
cpuUsage: [Function: cpuUsage],
dlopen: [Function: dlopen],
uptime: [Function: uptime],
memoryUsage: [Function: memoryUsage],
binding: [Function: binding],
_linkedBinding: [Function: _linkedBinding],
_events:
  { warning: [Function],
    newListener: [Function],
    removeListener: [Function],
    SIGWINCH: [Array] },
_rawDebug: [Function],
_eventsCount: 4,
domain: [Getter/Setter],
_maxListeners: undefined,
_fatalException: [Function],
_exiting: false,
assert: [Function],
config: { target_defaults: [Object], variables: [Object] },

```

```

    emitWarning: [Function],
    nextTick: [Function: nextTick],
    _tickCallback: [Function: _tickDomainCallback],
    _tickDomainCallback: [Function: _tickDomainCallback],
    stdout: [Getter],
    stderr: [Getter],
    stdin: [Getter],
    openStdin: [Function],
    exit: [Function],
    kill: [Function],
    argv0: 'D:\\SDK\\NodeJS\\node.exe' },
  Buffer:
    { [Function: Buffer]
      poolSize: 8192,
      from: [Function],
      alloc: [Function],
      allocUnsafe: [Function],
      allocUnsafeSlow: [Function],
      isBuffer: [Function: isBuffer],
      compare: [Function: compare],
      isEncoding: [Function],
      concat: [Function],
      byteLength: [Function: byteLength],
      [Symbol(node.isEncoding)]: [Function] },
  clearImmediate: [Function],
  clearInterval: [Function],
  clearTimeout: [Function],
  setImmediate: { [Function: setImmediate] [Symbol(util.promisify.custom)]: [Function] },
  setInterval: [Function],
  setTimeout: { [Function: setTimeout] [Symbol(util.promisify.custom)]: [Function] },
  module:
    Module {
      id: '<repl>',
      exports: {},
      parent: undefined,
      filename: null,
      loaded: false,
      children: [],
      paths:
        [ 'D:\\Projects\\NodeJS\\Panduan Node.js 8.x dalam 14 Bab\\Panduan Node.js 8.x dalam
14 Bab\\bab-5\\repl\\node_modules',
          'D:\\Projects\\NodeJS\\Panduan Node.js 8.x dalam 14 Bab\\Panduan Node.js 8.x dalam
14 Bab\\bab-5\\node_modules',
          'D:\\Projects\\NodeJS\\Panduan Node.js 8.x dalam 14 Bab\\Panduan Node.js 8.x dalam
14 Bab\\node_modules',
          'D:\\Projects\\NodeJS\\Panduan Node.js 8.x dalam 14 Bab\\node_modules',
          'D:\\Projects\\NodeJS\\node_modules',
          'D:\\Projects\\node_modules',
          'D:\\node_modules',
          'C:\\Users\\laptop.baru.bagus\\.node_modules',
          'C:\\Users\\laptop.baru.bagus\\.node_modules',
          'D:\\SDK\\NodeJS\\lib\\node' ] },

```

```
require:
{ [Function: require]
  resolve: { [Function: resolve] paths: [Function: paths] },
  main: undefined,
  extensions: { '.js': [Function], '.json': [Function], '.node': [Function] },
  cache: {} } }
```

Bab 6. Express.js

Pada pembahasan sebelumnya, kita telah mengetahui bahwa Node.js memiliki modul yang membuat Node.js menjadi web server.

Modul HTTP misalnya.

Akan tetapi, ada kalanya kita mungkin membutuhkan akses ke file yang statis melalui web server.

Atau mungkin juga kita ingin membuat routing pada banyak halaman.

Dengan hanya mengandalkan modul tadi, proses ini menjadi semakin rumit.

Oleh karena itu, developer mengembangkan beberapa framework untuk keperluan web bagi Node.js.

Framework-framework tersebut adalah:

- Express.js
- Hapi.js
- Mojito
- Meteor
- Derby
- Sails.js
- Koa.js
- Total.js

Karena banyak, maka saya tidak akan membahas semua framework tersebut.

Yang saya akan bahas adalah Express.js karena framework ini cukup umum digunakan.

Meng-install dan Menggunakan Express.js

Express.js adalah modul luar.

Oleh karena itu, kita harus menginstallnya terlebih dahulu dengan NPM sebelum digunakan di dalam kode kita.

Pertama-tama, kita akan membuat project Node.js terlebih dahulu.

Lakukan dengan cara yang sama seperti pembahasan sebelumnya.

Setelah itu, install Express.js dengan cara ini:

```
npm install express --save
```

Dengan parameter `--save`, maka Express.js didaftarkan di bagian "dependencies" dalam file "package.json".

Sekarang kita coba Express.js.

Caranya, buat file baru bernama "bab-6-testing-expressjs.js", kemudian isi dengan:

```
const express = require('express');
const app = express();
app.get('/', function (req, res) {
  res.send('Salam dari Express.js!');
});
app.listen(3000, function () {
  console.log('Aplikasi ini berjalan pada port 3000!');
});
```

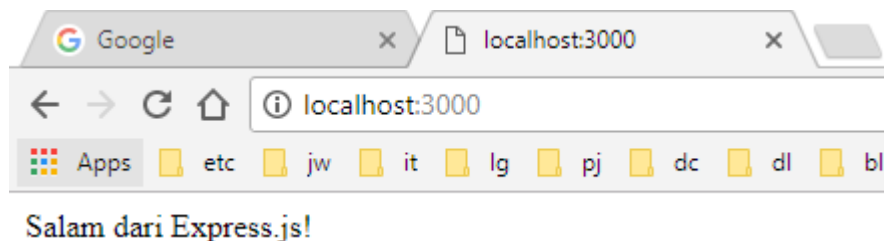
Kemudian, jalankan script tersebut dengan perintah:

```
node bab-6-testing-expressjs.js
```

Hasilnya dapat dilihat dengan membuka browser ke:

```
http://localhost:3000
```

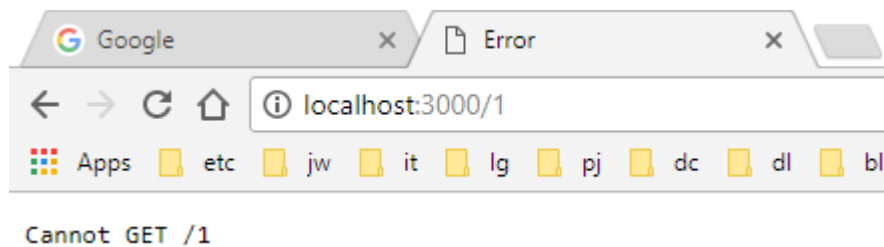
Outputnya seperti ini:



Apabila kita memasukkan path yang salah semisal:

```
http://localhost:3000/1
```

Maka browser akan menampilkan:



Hal itu disebabkan kita belum mendaftarkan route ke path `"/1"`.

Bagian ini:

```
app.listen(3000, function () {  
  console.log('Aplikasi ini berjalan pada port 3000!');  
});
```

Menyatakan bahwa Express.js menerima request di port 3000.

Express.js Routing

Express.js menangani routing get seperti yang barusan kita praktikkan.

Selain itu ada post, put, head, delete, options, trace, copy, lock, mkcol, move, purge, propfind, proppatch, unlock, report, mkactivity, checkout, merge, m-search, notify, subscribe, unsubscribe, patch, search, dan connect.

Penggunaan routing di atas serupa dengan get, yakni:

```
app.METHOD(PATH, HANDLER)
```

Di mana METHOD misalnya get, put, head, dan lain-lain.

Di mana PATH misalnya `"/"`, `"/1"`, `"/about"` dan lain-lain.

Dan di mana HANDLER adalah response yang disajikan dalam fungsi callback.

Ada pula suatu metode routing spesial dari Express.js yang bernama `"all"`.

Metode `"all"` menerima semua jenis request.

Jadi, jika kita melakukan ini:

```
app.all('/all', function (req, res) {  
  res.send('Express.js ALL!');  
});
```

Maka request get, post, put, delete dan yang lainnya akan dijawab dengan:

```
Express.js ALL!
```

Express.js Route Path

Tadi kita telah mencoba "/" sebagai Path.

Itu hanya satu contoh.

Sebenarnya masih banyak lagi Path lain yang bisa didefinisikan.

Seperti "/1", "/about" dan apapun.

Yang menarik, Path ini bisa diisi dengan regular expression.

Misalnya...

Ini valid untuk /path1/wyz atau /path1/wxyz:

```
app.get('/path1/wx?yz', function (req, res) {  
  res.send('wx?yz')  
});
```

Ini valid untuk /path2/wxyz, /path2/wxyz, /path2/wxxxxyz:

```
app.get('/path2/wx+yz', function (req, res) {  
  res.send('wx+yz')  
});
```

Ini valid untuk /path3/wxyz, /path3/wxayz, /path3/wxapapunyz, /path3/wx123yz:

```
app.get('/path3/wx*yz', function (req, res) {  
  res.send('wx*yz')  
});
```

Ini valid untuk /path4/wxe dan /path4/wxyze:

```
app.get('/path4/wx(yz)?e', function (req, res) {  
  res.send('wx(yz)?e')  
});
```

Route juga bisa merespon request dengan path yang memiliki parameter.

URL seperti ini:

```
http://localhost:3000/path5/books/100  
http://localhost:3000/path5/books/101  
http://localhost:3000/path5/books/102  
http://localhost:3000/path5/books/567
```

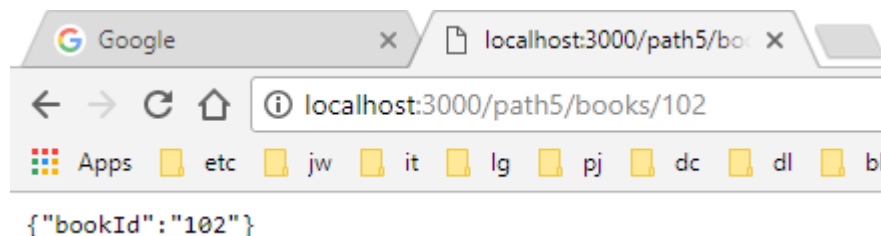
Akan diproses jika route-nya seperti ini:

```
app.get('/path5/books/:bookId', function (req, res) {  
  res.send(req.params)  
});  
  
app.listen(3000, function () {  
  console.log('Aplikasi ini berjalan pada port 3000!');  
});
```

Apabila ada request ke:

```
http://localhost:3000/path5/books/102
```

Maka isi dari req.params adalah:



Response Methods

Selain res.send, Express.js juga menyediakan beberapa metode respon lainnya. Berikut ini daftarnya:

- `res.download()`, untuk mendownload sesuatu.
- `res.end()`, mengakhiri proses response.
- `res.json()`, mengirim response berupa JSON.
- `res.jsonp()`, mengirim response berupa JSON dengan JSONP support.
- `res.redirect()`, melakukan redirect.
- `res.render()`, merender template.
- `res.sendFile()`, mengirim file dalam bentuk octet stream.
- `res.sendStatus()`, mengeset response status code dan mengirim representasi stringnya sebagai response body.

Middleware

Fungsi middleware adalah fungsi yang memiliki akses ke request object (req), response object (res), dan fungsi middleware selanjutnya dalam siklus request-response dari sebuah aplikasi.

Fungsi `next()` mengarahkan middleware ke middleware selanjutnya.

Agar memahaminya, buat file baru bernama "bab-6-middleware-next.js", kemudian isi dengan:

```
const express = require('express');
const app = express();
app.use(function (req, res, next) {
  console.log('Time:', Date.now());
  next();
});
app.use('/pengguna/:id', function (req, res, next) {
  console.log('Request Type:', req.method);
  next();
});
app.get('/', function (req, res) {
  console.log('Salam dari Express.js!');
  res.send('Salam dari Express.js!');
});

app.get('/pengguna/:id', function (req, res) {
  console.log('Menampilkan User ID');
  res.send('Menampilkan User ID');
});
app.listen(3000, function () {
  console.log('Berjalan di port 3000!');
});
```

Selanjutnya, jalankan dengan perintah ini:

```
node bab-6-middleware-next.js
```

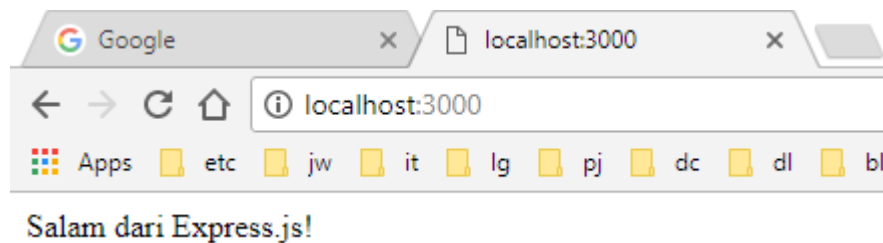
Kemudian buka browser ke:

```
http://localhost:3000
```

Nanti di **CONSOLE WINDOW (BUKAN DI BROWSER)**, akan tampil seperti ini:

```
Berjalan di port 3000!  
suatu bilangan: 123  
Salam dari Express.js!  
suatu bilangan: 123
```

Sementara di browser akan menampilkan:



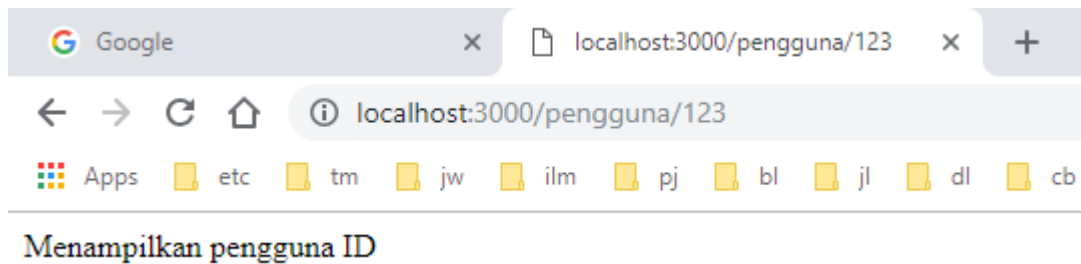
Selanjutnya coba browse ke:

```
http://localhost:3000/user/123
```

Nanti di console window akan tampil yang semacam ini:

```
Berjalan di port 3000!  
suatu bilangan: 123  
Salam dari Express.js!  
suatu bilangan: 123  
suatu bilangan: 123  
suatu bilangan: 123  
Request Type: GET  
Menampilkan pengguna ID
```

Tapi di browser akan tampil:



Kita lihat bahwa, dengan menggunakan use, kita bisa meng-hijack request yang dikirimkan.

Jika tidak terikat dengan path-nya, cukup lakukan ini:

```
app.use(function (req, res, next) {  
  console.log('suatu bilangan:', 123);  
  next();  
});
```

Sedangkan jika terikat dengan path lakukan ini:

```
app.use('/pengguna/:id', function (req, res, next) {  
  console.log('Request Type:', req.method)  
  next()  
})
```

Middleware digunakan juga untuk tujuan lain seperti untuk mengakses file statis.

Untuk mencobanya, buatlah folder bernama "public123" di folder "bab-6".

Kemudian buat script baru bernama "bab-6-middleware-static.js", dan isi dengan ini:

```
const express = require('express');  
const app = express();  
app.use(express.static('public123'));  
app.listen(3000, function () {  
  console.log('Aplikasi ini berjalan pada port 3000!');  
});
```

Nanti seisi folder public123 dapat diakses dengan cara ini:

```
http://localhost:3000/images/logo.png  
http://localhost:3000/css/bootstrap.min.css  
http://localhost:3000/js/jquery.min.js  
http://localhost:3000/home.html
```

Template Engine

Terdapat beberapa template engine untuk Express.js yang berfungsi untuk melakukan server render terhadap response html.

Dengan template engine tersebut, kita bisa membuat sebuah template html yang dapat diselipkan dengan data-data tertentu.

Dengan begitu, response html bisa terlihat dinamis.

Beberapa template engine yang populer adalah:

- Pug
- Mustache
- EJS

Akan tetapi, template engine tidak mutlak diperlukan.

Jika kita membuat single page web application, kita mungkin lebih memilih client render dengan menggunakan Angular maupun Vue.

Menggunakan Template Engine EJS

Sekarang, kita akan mencoba menggunakan EJS untuk melakukan render.

Pertama, buatlah folder baru di dalam folder "bab-6" bernama "views".

Folder itu adalah folder default untuk menyimpan template di EJS.

Selanjutnya di dalam folder "bab-6/views" buatlah file baru bernama "bab-6-home.ejs".

Isi file tersebut adalah seperti ini:

```
<!DOCTYPE html>  
<html lang="en">
```



```

<head>
<meta charset="utf-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1">

<title><%=judul%></title>
</head>
<body>
  <h1><%=judul%></h1>
  <p><%=pesan%></p>
</body>
</html>

```

Perhatikan bahwa <%=judul%> dan <%=pesan%> adalah variabel yang akan dihubungkan dengan Node.js di sisi server.

Sekarang, buatlah file baru di dalam folder "bab-6" dengan nama "bab-6-express-ejs.js".

Isinya seperti ini:

```

var express = require('express');
var app = express();

app.set('view engine', 'ejs');

app.get('/', function (req, res) {
  res.render('bab-6-home', {
    judul: "Judul ini Ditulis dengan EJS", //INI YANG DIHUBUNGAN DENGAN EJS
    pesan: "pesan ditulis dengan EJS"//INI YANG DIHUBUNGAN DENGAN EJS
  });
});

app.listen(3000, function () {
  console.log('Aplikasi ini berjalan pada port 3000!');
});

```

Perhatikan juga bahwa variabel judul di atas akan dihubungkan dengan <%=judul%> di file template EJS.

Sementara variabel pesan akan dihubungkan dengan <%=pesan%> di EJS.

Oleh karena itu, apapun yang ditulis pada variabel judul dan pesan di server,

efeknya akan terlihat di browser.

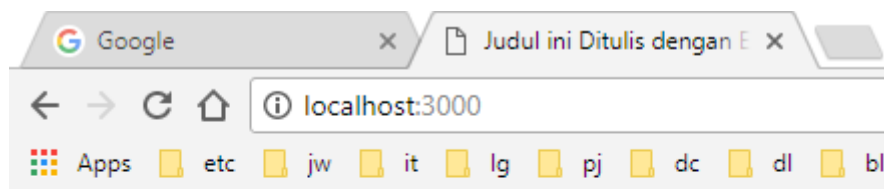
Sekarang, jalankan servernya:

```
node bab-6-express-ejs.js
```

Kemudian buka browser ke:

```
http://localhost:3000/
```

Outputnya akan seperti ini di browser:



Judul ini Ditulis dengan EJS

pesan ditulis dengan EJS

Bab 7. Database

Node.js memiliki modul luar yang sangat banyak dan jumlahnya semakin meningkat.

Di sisi lain, karena Node.js umumnya digunakan untuk aplikasi yang membutuhkan database, maka jumlah modul untuk manajemen database juga banyak.

Adapun beberapa database yang di-support oleh modul luar Node.js adalah:

- MySQL
- SQLite
- MongoDB
- NeDB
- PostgreSQL
- CouchDB
- Redis
- MariaDB
- Cassandra

Karena sangat banyak, maka kita akan membahas sebagian database saja, yakni MySQL, SQLite, dan MongoDB.

MySQL

Database MySQL merupakan salah satu database yang paling populer di kalangan web developer.

Jika kita belajar PHP, MySQL sering disertakan pada pelajaran tersebut.

Beberapa keunggulan MySQL adalah:

- Support yang bagus karena dukungan komunitas
- Mudah di-install
- Murah dan bisa didapatkan dengan gratis
- Merupakan industry standard

Beberapa kelemahan MySQL adalah:

- Memiliki sedikit masalah dalam stabilitas
- Tidak terlalu open source
- Dimiliki oleh perusahaan dan bukan community driven
- Terlalu tergantung pada add-ons

Meng-install MySQL

Untuk menggunakan MySQL dengan Node.js, kita harus meng-install MySQL server terlebih dahulu.

Download-lah MySQL Community Edition di:

```
https://www.mysql.com/downloads/
```

Kemudian install software tersebut.

Setelah meng-install-nya, install MySQL driver untuk Node.js dengan cara:

```
npm install mysql --save
```

Membuka Koneksi ke MySQL

Untuk membuka koneksi ke MySQL (Menurut W3Schools):

```
var mysql = require('mysql'); //import module mysql

var connection = mysql.createConnection({
  host: "domain_atau_ip",
  user: "username_anda",
  password: "password_anda"
});

connection.connect(function(gag) {
  if (gag) {
    console.error('error buka koneksi: ' + gag.stack);
    return;
  }
  console.log('terkoneksi dengan id ' + connection.threadId);
});
```

Untuk menjalankannya, jalankan server MySQL terlebih dahulu, lalu jalankan script di atas.

Output-nya (jika host, user, dan password benar):

```
terkoneksi dengan id
```

Fungsi Query pada Node.js MySQL

Untuk membuat database, membuat tabel, melakukan insert dan select, semua dapat dilakukan dengan fungsi query:

```
connection.query("query_nya_di_sini", function (gag, result) {  
  if (gag){  
    throw gag;  
  }  
  console.log("Result: " + result);  
});
```

Jadi, apabila ingin melakukan create database (Menurut W3Schools):

```
var mysql = require('mysql');  
  
var connection = mysql.createConnection({  
  host: "domain_atau_ip",  
  user: "username_anda",  
  password: "password_anda"  
});  
  
connection.connect(function(gag) {  
  if (gag){  
    throw gag;  
  }  
  console.log("Terkoneksi dengan sukses!");  
  connection.query("CREATE DATABASE puppies", function (gag, result) {  
    if (gag){  
      throw gag;  
    }  
    console.log("Database telah dibuat");  
    connection.destroy(); //PERHATIKAN YANG INI!  
  });  
});
```

Perhatikan bagian:

```
connection.destroy();
```

Method ini dipanggil karena kita ingin menghentikan eksekusi setelah query berhasil dijalankan.

Jika method tersebut tidak dipanggil, program akan tetap berjalan.

Apabila ingin membuat tabel (Menurut W3Schools):

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

var connection = mysql.createConnection({
  host: "domain_atau_ip",
  user: "username_anda",
  password: "password_anda",
  database: "puppies"
});

connection.connect(function(gag) {
  if (gag){
    throw gag;
  }
  console.log("Terkoneksi dengan sukses!");
  connection.query("CREATE TABLE books (title VARCHAR(255), description VARCHAR(255))", function (gag, result) {
    if (gag){
      throw gag;
    }
    console.log("Table selesai dibuat");
    connection.destroy();
  });
});
```

Perhatikan bahwa kita harus memilih terlebih dahulu database mana yang ingin dibuat tabelnya pada createConnection:

```
database: "nama_database"
```

Apabila ingin melakukan insert (Menurut W3Schools):

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

var connection = mysql.createConnection({
  host: "domain_atau_ip",
  user: "username_anda",
  password: "password_anda",
```

```

        database: "puppies"
    });

    connection.connect(function(gag) {
        if (gag){
            throw gag;
        }
        console.log("Terkoneksi dengan sukses!");

        connection.query("INSERT INTO books (title, description) VALUES ('Cara Membuat
Kopi', 'bagaimana caranya membuat kopi')", function (gag, result) {
            if (gag){
                throw gag;
            }
            console.log("1 record diinsert");
            connection.destroy();
        });
    });
});

```

Apabila ingin melakukan select (Menurut W3Schools):

```

var mysql = require('mysql');

var connection = mysql.createConnection({
    host: "domain_atau_ip",
    user: "username_anda",
    password: "password_anda",
    database: "puppies"
});

connection.connect(function(gag) {
    if (gag){
        throw gag;
    }
    console.log("Terkoneksi dengan sukses!");
    connection.query("SELECT * FROM books", function (gag, result, fields) {
        if (gag){
            throw gag;
        }
        console.log(result);
        connection.destroy();
    });
});

```

```
});
```

SQLite

Tidak kalah populernya dengan MySQL, SQLite yang merupakan versi serverless (tanpa server) dari SQL sangat banyak digunakan pada aplikasi Android.

Bahkan, di Android SDK, SQLite merupakan database resmi dari aplikasinya.

Beberapa keunggulan SQLite adalah:

- Tidak memerlukan server.
- Tidak memerlukan konfigurasi yang rumit.
- Dapat digunakan pada berbagai OS.
- Mudah dipackage.
- Tersedia database browsernya, seperti DB Browser for SQLite.

Beberapa kelemahan SQLite adalah:

- Performance tidak terlalu bagus untuk aplikasi client/server.
- Tidak cocok untuk website yang banyak melakukan write.
- Database hanya dapat disimpan pada single disk.

Meng-install SQLite

Untuk menggunakan SQLite di Node.js, kita tidak perlu meng-install servernya terlebih dahulu, karena SQLite memang serverless.

Akan tetapi, kita harus meng-install module Node.js untuk SQLite terlebih dahulu dengan cara:

```
npm install sqlite3 --save
```

Fungsi Run dan Prepare pada Node SQLite3

SQLite memiliki query yang kurang lebih sama dengan MySQL.

Akan tetapi, pada API Node.js-nya terdapat beberapa perbedaan.

Salah satunya adalah pada MySQL fungsi querynya adalah query, sedangkan pada SQLite, fungsi querynya adalah run.

Jadi apabila kita ingin membuat tabel dengan SQLite:


```
var sqlite3 = require('sqlite3').verbose();

var db = new sqlite3.Database('nama_database.db');

db.serialize(function() {
  db.run("CREATE TABLE if not exists macan (info TEXT)");
});
```

Apabila kita ingin memasukkan beberapa data sekaligus, kita bisa menggunakan fungsi prepare.

Misalnya, pada tabel books tadi, kita ingin memasukkan beberapa row info.

Untuk melakukannya, begini caranya:

```
var sqlite3 = require('sqlite3').verbose();

var db = new sqlite3.Database('nama_database.db');

var prepared = db.prepare("INSERT INTO macan VALUES (?)");

for (var i = 0; i < 10; i++) {
  prepared.run("Info " + i);
}

prepared.finalize();
```

Tanda tanya pada fungsi prepare tadi adalah parameter yang diikat pada variabel prepared.

Jadi tanda tanya tadi di-replace dengan "Info 1", "Info 2", dan seterusnya.

Adapun fungsi finalize adalah untuk mengakhiri statement.

Fungsi finalize tidak mutlak diperlukan, kecuali ada delay yang cukup lama untuk menjalankan query selanjutnya.

Fungsi Each

Untuk mendapatkan row dari SQLite, digunakan fungsi each:

```
var sqlite3 = require('sqlite3').verbose();
```

```
var db = new sqlite3.Database('nama_database.db');  
  
db.each("SELECT rowid AS id, info FROM macan", function(err, row) {  
    console.log(row.info);  
});
```

Dengan demikian, isi dari tabel books akan ditampilkan.

Bab 8. Websocket

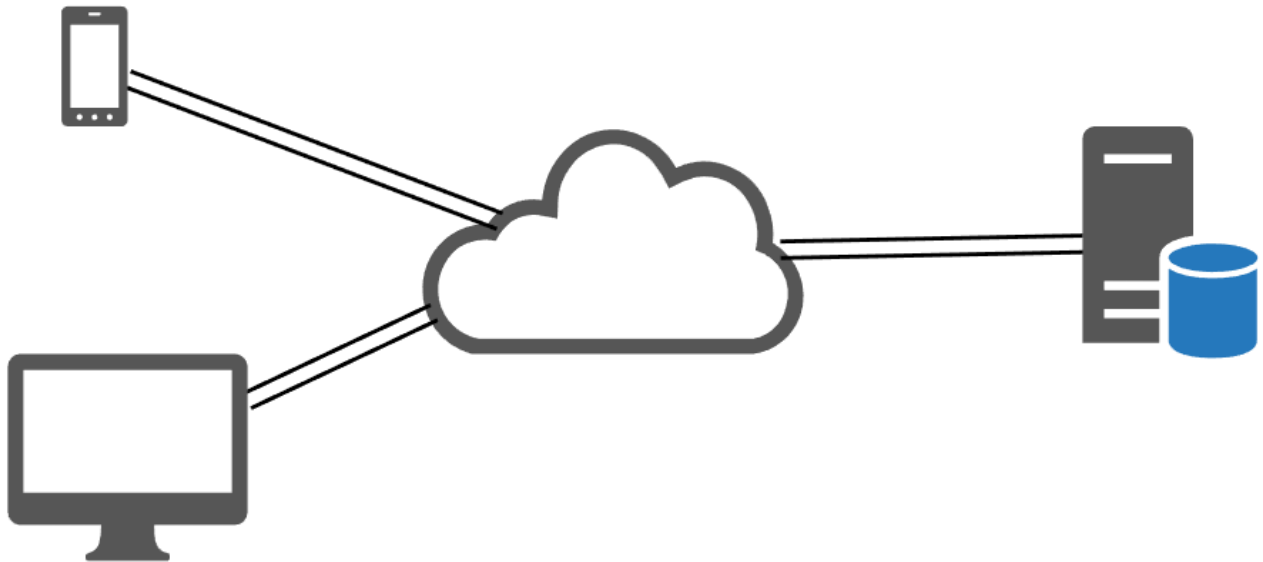
Apa Itu Websocket?

Websocket adalah protokol komunikasi dua arah antara server dan browser.

Protokol komunikasi ini bersifat bi-directional dan full duplex.

Artinya, kedua pihak berkomunikasi pada waktu yang sama.

Karena koneksinya bersifat persisten, server dapat mengirim data ke browser.



Websocket

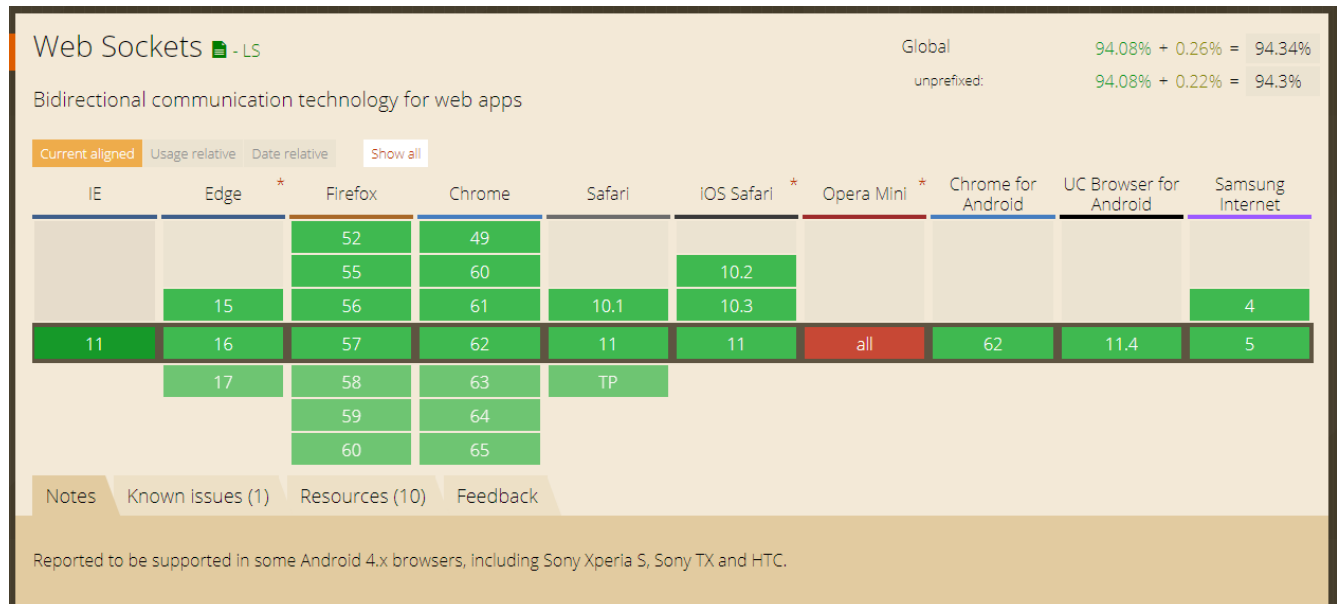
Walaupun demikian, koneksi harus dimulai dari client terlebih dahulu karena server tidak bisa memulai koneksi ke client.

Selain untuk browser, websocket juga dapat diterapkan pada Internet of Things (IoT).

Akibatnya, perangkat IoT yang terkoneksi ke server bisa berkomunikasi secara bi-directional dan full duplex seperti browser.

Saat ini, sebagian browser, baik untuk desktop maupun Android sudah mendukung websocket.

Kita bisa melihatnya di gambar ini.



Browser yang Mendukung Websocket (<https://caniuse.com/#search=websocket>)

Apa Bedanya dengan AJAX?

Walaupun fungsinya bisa sama, ada beberapa perbedaan antara Websocket dengan AJAX.

Perbedaan itu dirangkum dalam tabel ini.

	Websocket	AJAX
Koneksi	Sekali request, akan tetap terkoneksi.	Harus tetap me-request agar tetap terkoneksi.
Pengiriman Data	Dari client ke server dan sebaliknya.	Dari client ke server saja.
Bandwidth	Hemat bandwidth karena ukuran data frame kecil	Kurang hemat bandwidth karena ukuran data frame lebih besar.

Membuat Aplikasi Websocket Pertama

Agar lebih jelas dalam memahami Websocket, kita akan mencobanya dengan Node.js.

Pertama-tama, buat folder bernama "bab-8".

Selanjutnya buka PowerShell/Terminal pada folder tersebut dan buatlah project Node.js dengan nama "bab-8".

Kemudian install modul Websocket yang bernama "ws":

```
npm install ws --save
```

Pastikan juga bahwa browser yang kita gunakan mendukung Websocket.

Kita bisa melihat gambar "Browser yang Mendukung Websocket" pada pembahasan sebelumnya.

Selanjutnya, buat file baru pada folder "bab-8" dengan nama "bab-8-websocket-pertama.js", kemudian isi dengan kode ini (menurut WebSocket):

```
const WebSocket = require('ws');

const wss = new WebSocket.Server({
  port: 9397
});

wss.on('connection', function koneksi(ws) {
  ws.on('message', function pesanDatang(pesan) {
    console.log('Diterima: %s', pesan);
  });

  ws.send('pesan dikirimkan ke client');
});
```

Selanjutnya, buat file baru bernama "bab-8-websocket-pertama.html", kemudian isi dengan kode ini:

```
<!DOCTYPE html>
<html>

<head>
```

```

<style>
    div {
        display: inline;
    }
</style>
<script>
    var host = 'localhost';
    var ws = new WebSocket('ws://' + host + ':9397');

    ws.onopen = function (event) {
        ws.send("pesan dikirimkan ke server");
    };

    ws.onmessage = function (event) {
        document.getElementById('pesan').innerHTML = event.data;
    };
</script>
</head>

<body>
    <h3>Percobaan WebSocket</h3><br>
    Pesan: <div id='pesan'></div><br>
</body>

</html>

```

Sekarang simpan kedua file tersebut dan buka PowerShell/Terminal pada folder "bab-8".

Kemudian jalankan server-nya:

```
node bab-8-websocket-pertama.js
```

Selanjutnya, buka file "bab-8-websocket-pertama.html" sebagai file html dengan browser kita.

Output pada browser akan seperti ini:

```
Pesan: pesan dikirimkan ke client
```

Output pada PowerShell/Terminal akan seperti ini:

```
Diterima: pesan dikirimkan ke server
```

Kode Websocket yang "bab-8-websocket-pertama.js" adalah server-nya, sedangkan yang html adalah client-nya.

Di situ tampak bahwa server bisa mengirimkan data ke client pada bagian kode yang ini:

```
//Kode ini di sisi server
wss.on('connection', function koneksi(ws) {
  ws.on('message', function pesanDatang(pesan) {
    console.log('Diterima: %s', pesan);
  });

  ws.send('pesan dikirimkan ke client');
});
```

Event "message" terjadi ketika server menerima data dari client.

Event-Event pada Websocket

Di sisi client, Websocket API pada browser memiliki 4 event utama.

Event-event tersebut adalah:

- Open
- Message
- Close
- Error

Event Open terjadi ketika koneksi antara client dan server terbangun.

Pada contoh sebelumnya, kita mendeteksi event ini dengan:

```
//Kode ini di sisi client
ws.onopen = function(event) { //YANG INI
  ws.send("pesan dikirimkan ke server");
};
```

Jadi, event Open dapat diterima dengan callback WebSocket.onopen di sisi browser.

Event Message terjadi setiap kali server mengirim data (dan client menerima data dari server).

Data yang bisa dikirim dari server adalah teks dan binary.

```
ws.onmessage = function(event){  
    //Kode untuk menangani data  
};
```

Event Close terjadi pada saat koneksi diakhiri.

Setelah event Close terjadi, maka tidak ada lagi data yang bisa ditransfer antara client dan server.

```
ws.onclose = function(event){  
    //Kode untuk menangani data  
};
```

Event Error terjadi ketika ada kesalahan yang terjadi dalam komunikasi.

Event ini selalu diikuti dengan diakhirinya koneksi.

```
ws.onerror = function(event){  
    //Kode untuk menangani data  
};
```

Agar lebih paham dengan event-event WebSocket, kita akan membuat sebuah aplikasi sederhana.

Pertama, buka folder "bab-8" dan buatlah sebuah file bernama "bab-8-websocket-events.js", kemudian isi dengan kode ini (menurut WebSocket):

```
const WebSocket = require('ws');  
  
const wss = new WebSocket.Server({  
    port: 9876  
});  
  
wss.on('connection', function koneksi(ws) {  
    ws.on('message', function pesanDatang(pesan) {  
        console.log('Diterima: %s', pesan);  
        ws.send('server mendeteksi tombol Kirim Data diklik');  
    });  
});
```


Kemudian, pada folder yang sama, buatlah sebuah file bernama "bab-8-websocket-events.html" lalu isi dengan kode ini (bila agak sulit membacanya, silakan lihat source code yang disertakan dengan buku ini):

```
<!DOCTYPE html>
<html>

<head>
  <style>
    div {
      display: inline;
    }
  </style>
  <script>
    var host = 'localhost';
    var ws = new WebSocket('ws://' + host + ':9876');

    function myLog(theType) {
      var elm = document.createElement('p');
      elm.innerHTML = theType;
      return elm;
    }

    ws.onopen = function (event) {
      document.getElementById('logs').appendChild(myLog("onopen"));
    };

    ws.onmessage = function (event) {
      document.getElementById('logs').appendChild(myLog("onmessage: " +
event.data));
    };

    ws.onclose = function (event) {
      var reason;
      // Kode bisa dilihat di http://tools.ietf.org/html/rfc6455#section-7.4.1
      if (event.code == 1000)
        reason = event.code + ": " +
          "Normal closure, meaning that the purpose for which the
connection was established has been fulfilled.";
      else if (event.code == 1001)
        reason = event.code + ": " +
```

```

        "An endpoint is \"going away\", such as a server going down
or a browser having navigated away from a page.";
    else if (event.code == 1002)
        reason = event.code + ": " + "An endpoint is terminating the
connection due to a protocol error";
    else if (event.code == 1003)
        reason = event.code + ": " +
        "An endpoint is terminating the connection because it has
received a type of data it cannot accept (e.g., an endpoint that understands only
text data MAY send this if it receives a binary message).";
    else if (event.code == 1004)
        reason = event.code + ": " + "Reserved. The specific meaning
might be defined in the future.";
    else if (event.code == 1005)
        reason = event.code + ": " + "No status code was actually
present.";
    else if (event.code == 1006)
        reason = event.code + ": " +
        "The connection was closed abnormally, e.g., without sending
or receiving a Close control frame";
    else if (event.code == 1007)
        reason = event.code + ": " +
        "An endpoint is terminating the connection because it has
received data within a message that was not consistent with the type of the message
(e.g., non-UTF-8 [http://tools.ietf.org/html/rfc3629] data within a text message).";
    else if (event.code == 1008)
        reason = event.code + ": " +
        "An endpoint is terminating the connection because it has
received a message that \"violates its policy\". This reason is given either if there
is no other suitable reason, or if there is a need to hide specific details about the
policy.";
    else if (event.code == 1009)
        reason = event.code + ": " +
        "An endpoint is terminating the connection because it has
received a message that is too big for it to process.";
    else if (event.code ==
        1010
    ) // Note that this status code is not used by the server, because
it can fail the WebSocket handshake instead.
        reason = event.code + ": " +
        "An endpoint (client) is terminating the connection because
it has expected the server to negotiate one or more extension, but the server didn't
return them in the response message of the WebSocket handshake. <br /> Specifically,

```

```

the extensions that are needed are: " +
    event.reason;
else if (event.code == 1011)
    reason = event.code + ": " +
        "A server is terminating the connection because it
encountered an unexpected condition that prevented it from fulfilling the request.";
else if (event.code == 1015)
    reason = event.code + ": " +
        "The connection was closed due to a failure to perform a TLS
handshake (e.g., the server certificate can't be verified).";
else
    reason = event.code + ": " + "Unknown reason";

    document.getElementById('logs').appendChild(myLog("onclose: " +
reason));
};

ws.onerror = function (event) {
    document.getElementById('logs').appendChild(myLog("onerror"));
};

function kirimData() {
    ws.send("pesan dikirimkan ke server");
}

function tutupKoneksi() {
    ws.close();
}
</script>
</head>

<body>
    <h3>Percobaan WebSocket Events</h3>
    <br />
    <button id='btn-send' onclick="kirimData()">Kirim Data ke Server</button>
    <button id='btn-close' onclick="tutupKoneksi()">Tutup Koneksi</button>
    <div id='logs'></div>
</body>

</html>

```

Sekarang, tanpa menjalankan server, buka file html tadi dengan browser.

Nanti akan muncul pesan ini:

```
onerror  
  
onclose: 1006: The connection was closed abnormally, e.g., without sending or receiving a  
Close control frame
```

Hal itu disebabkan server tidak bisa dihubungi.

Kode 1006 menjelaskan sebabnya.

Kode ini, entah kenapa, justru ada pada event onclose, bukan pada onerror.

Tapi karena semua error akan diikuti close, maka gunakan cara ini.

Kode-kode onclose selengkapnya bisa didapatkan di:

```
http://tools.ietf.org/html/rfc6455#section-7.4.1
```

Sekarang, coba jalankan server dan refresh browser kita:

```
node bab-8-websocket-events.js
```

Maka akan muncul output seperti ini:

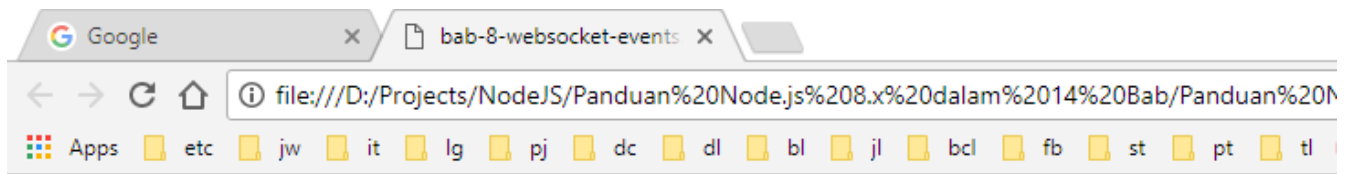
```
onopen
```

Sementara itu, jika tombol "Kirim Data ke Server", maka client akan menerima pesan balasan dari server:

```
onmessage: server mendeteksi tombol Kirim Data diklik
```

Jika kita tetap pada halaman tersebut tanpa me-refreshnya, kemudian kita hentikan server dengan menekan tombol CTRL+C di PowerShell/Terminal, maka tidak lama kemudian akan muncul pesan ini di browser:

```
onclose: 1006: The connection was closed abnormally, e.g., without sending or receiving a  
Close control frame
```



Percobaan WebSocket Events

Kirim Data ke Server

Tutup Koneksi

onopen

onmessage: server mendeteksi tombol Kirim Data diklik

onclose: 1006: The connection was closed abnormally, e.g., without sending or receiving a Close control frame

Kali ini tanpa onerror.

Jika kita mengakhiri koneksi dengan menekan tombol "Tutup Koneksi", maka kita akan mendapat output ini di browser:

```
onclose: 1000: Normal closure, meaning that the purpose for which the connection was established has been fulfilled.
```

Artinya, koneksi diputus dengan normal.

Broadcast Data dengan WebSocket

Ada kalanya kita ingin menyebarkan pesan dari server ke semua client yang terkoneksi.

Atau bisa disebut juga dengan broadcast.

Pada penerapannya di dunia nyata, mungkin ada sebuah aplikasi chat yang menyediakan fitur public chatroom seperti di Yahoo Messenger di masa lalu, atau mungkin juga sebuah shoutbox yang di-embed dalam sebuah blog.

Broadcast diperlukan untuk aplikasi semacam itu.

Untuk melakukannya dengan WebSocket, cara kerjanya cukup sederhana.

Kita cukup melakukan iterasi pada semua client yang terkoneksi pada server WebSocket, kemudian mengirim pesan broadcast kita kepada setiap client.

Untuk melakukan itu, buat sebuah file baru bernama "bab-8-websocket-broadcast.js" pada folder "bab-8", kemudian isi dengan kode ini (menurut

WebSocket):

```
const WebSocket = require('ws');

const wss = new WebSocket.Server({
  port: 9876
});

wss.on('connection', function koneksi(ws) {
  ws.on('message', function pesanDatang(data) {
    wss.clients.forEach(function each(item) {
      if (item !== ws && item.readyState === WebSocket.OPEN) {
        console.log(data);
        item.send(data);
      }
    });
  });
});
```

Selanjutnya, bagian client, buat file baru bernama "bab-8-websocket-broadcast.html", kemudian isi dengan kode ini (kodenya sama persis dengan "bab-8-websocket-events.html", kita tinggal copy-paste saja):

```
<!DOCTYPE html>
<html>

<head>
    <style>
        div {
            display: inline;
        }
    </style>
    <script>
        var host = 'localhost';
        var ws = new WebSocket('ws://' + host + ':9876');

        function myLog(theType) {
            var elm = document.createElement('p');
            elm.innerHTML = theType;
            return elm;
        }
    </script>

```

```

ws.onopen = function (event) {
    document.getElementById('logs').appendChild(myLog("onopen"));
};

ws.onmessage = function (event) {
    document.getElementById('logs').appendChild(myLog("onmessage: " +
event.data));
};

ws.onclose = function (event) {
    var reason;
    // Kode bisa dilihat di http://tools.ietf.org/html/rfc6455#section-7.4.1
    if (event.code == 1000)
        reason = event.code + ": " +
            "Normal closure, meaning that the purpose for which the
connection was established has been fulfilled.";
    else if (event.code == 1001)
        reason = event.code + ": " +
            "An endpoint is \"going away\", such as a server going down
or a browser having navigated away from a page.";
    else if (event.code == 1002)
        reason = event.code + ": " + "An endpoint is terminating the
connection due to a protocol error";
    else if (event.code == 1003)
        reason = event.code + ": " +
            "An endpoint is terminating the connection because it has
received a type of data it cannot accept (e.g., an endpoint that understands only
text data MAY send this if it receives a binary message).";
    else if (event.code == 1004)
        reason = event.code + ": " + "Reserved. The specific meaning
might be defined in the future.";
    else if (event.code == 1005)
        reason = event.code + ": " + "No status code was actually
present.";
    else if (event.code == 1006)
        reason = event.code + ": " +
            "The connection was closed abnormally, e.g., without sending
or receiving a Close control frame";
    else if (event.code == 1007)
        reason = event.code + ": " +
            "An endpoint is terminating the connection because it has

```

```

received data within a message that was not consistent with the type of the message
(e.g., non-UTF-8 [http://tools.ietf.org/html/rfc3629] data within a text message).";
    else if (event.code == 1008)
        reason = event.code + ": " +
            "An endpoint is terminating the connection because it has
received a message that \"violates its policy\". This reason is given either if there
is no other suitable reason, or if there is a need to hide specific details about the
policy.";

        else if (event.code == 1009)
            reason = event.code + ": " +
                "An endpoint is terminating the connection because it has
received a message that is too big for it to process.";
        else if (event.code ==
            1010
        ) // Note that this status code is not used by the server, because
it can fail the WebSocket handshake instead.
            reason = event.code + ": " +
                "An endpoint (client) is terminating the connection because
it has expected the server to negotiate one or more extension, but the server didn't
return them in the response message of the WebSocket handshake. <br /> Specifically,
the extensions that are needed are: " +
                event.reason;
        else if (event.code == 1011)
            reason = event.code + ": " +
                "A server is terminating the connection because it
encountered an unexpected condition that prevented it from fulfilling the request.";
        else if (event.code == 1015)
            reason = event.code + ": " +
                "The connection was closed due to a failure to perform a TLS
handshake (e.g., the server certificate can't be verified).";
        else
            reason = event.code + ": " + "Unknown reason";

        document.getElementById('logs').appendChild(myLog("onclose: " +
reason));
    };

    ws.onerror = function (event) {
        document.getElementById('logs').appendChild(myLog("onerror"));
    };

    function kirimData() {
        ws.send("pesan dikirimkan ke server");
    }

```



```

    }

    function tutupKoneksi() {
        ws.close();
    }
</script>
</head>

<body>
    <h3>Percobaan WebSocket Events</h3>
    <br />
    <button id='btn-send' onclick=' kirimData()'>Kirim Data ke Server</button>
    <button id='btn-close' onclick='tutupKoneksi()'>Tutup Koneksi</button>
    <div id='logs'></div>
</body>

</html>

```

Sekarang, jalankan server-nya dengan PowerShell/Terminal:

```
node bab-8-websocket-broadcast.js
```

Kemudian buka "bab-8-websocket-broadcast.html" pada 3 tab browser.

Pada tab yang paling kiri, tekan tombol "Kirim Data ke Server".

Output pada tab ini akan kosong.

Tapi lihat pada kedua tab lainnya.

Di kedua tab lainnya, yang tengah dan kanan, akan muncul output seperti ini:

```
onmessage: pesan dikirimkan ke server
```

Ini menunjukkan bahwa semua pesan di-broadcast ke seluruh client, kecuali ke browser pengirim.

Sejalan dengan kode ini:

```

wss.clients.forEach(function each(item) {
    if (item !== ws && item.readyState === WebSocket.OPEN) {
        console.log(data);
        item.send(data);
    }
});

```

```
    }  
  });
```

Bahwa setiap client yang bukan pengirim akan dikirimkan data.

Mendapatkan IP Address Client dengan Websocket

Kita bisa mendapatkan IP address client dengan menggunakan argument ke-2 dari connection callback.

Untuk mencobanya, buat file bernama "bab-8-websocket-client-address.js" pada folder "bab-8", kemudian isi dengan kode ini (menurut WebSocket):

```
const WebSocket = require('ws');  
  
const wss = new WebSocket.Server({  
  port: 9876  
});  
  
wss.on('connection', function koneksi(ws, req) {  
  const ip = req.connection.remoteAddress;  
  console.log(ip);  
  ws.send(ip);  
});
```

Selanjutnya, buat file bernama "bab-8-websocket-client-address.html" pada folder "bab-8", kemudian isi dengan kode ini:

```
<!DOCTYPE html>  
<html>  
  
<head>  
  <style>  
    div {  
      display: inline;  
    }  
  </style>  
  <script>  
    var host = 'localhost';  
    var ws = new WebSocket('ws://' + host + ':9876');  
  
    ws.onmessage = function (event) {
```

```

        document.getElementById('sang-ip').innerHTML = event.data;
    };
</script>
</head>

<body>
    <h3>IP Address Client dengan WebSocket</h3><br>
    IP: <div id='sang-ip'></div><br>
</body>

</html>

```

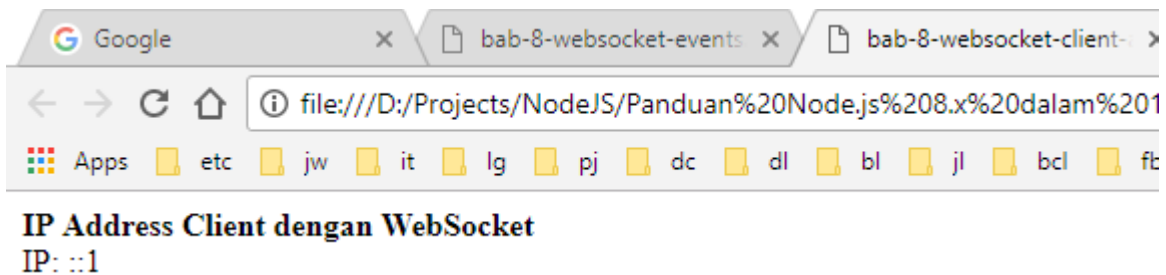
Jalankan server dengan PowerShell/Terminal:

```
node bab-8-websocket-client-address.js
```

Kemudian, buka file "bab-8-websocket-client-address.html" dengan browser.

Output di browser akan seperti ini:

```
IP: ::1
```



Sedangkan output pada PowerShell/Terminal seperti ini;

```
::1
```

```
Windows PowerShell
PS D:\Projects\NodeJS\Panduan Node.js 8.x dalam 14 Bab\Panduan Node.js 8.x dalam 14 Bab\bab-8> node bab-8-websocket-event-ts
Diterima: pesan dikirimkan ke server
PS D:\Projects\NodeJS\Panduan Node.js 8.x dalam 14 Bab\Panduan Node.js 8.x dalam 14 Bab\bab-8> node bab-8-websocket-client-address.js
::1
```

Menggunakan Socket.io dengan Express.js

Kita juga bisa menggunakan Websocket bersama Express.js.

Prinsipnya, kita menggunakan server yang didapat dari `createServer` dari modul HTTP sebagai input bagi `WebSocket.Server`:

```
const server = http.createServer(app);
const wss = new WebSocket.Server({ server }); //MASUKKAN KE SINI
```

Akan tetapi, karena tidak ada jaminan bahwa semua browser mendukung Websocket, sebaiknya kita menggunakan sebuah framework.

Framework yang bisa menjamin bahwa komunikasi antara client dan server bekerja walaupun browser yang kita gunakan tidak mendukung Websocket.

Dengan framework tersebut, apabila browser tidak mendukung Websocket, maka framework tersebut akan menggunakan protokol lain.

Framework itu adalah Socket.io.

Karena kita bermaksud menggunakan Express.js yang merupakan framework, maka lebih baik sekalian saja menggunakan framework Socket.io untuk Websocket-nya.

Untuk menghubungkan Socket.io dengan Express.js, pastikan kita telah menginstall kedua modul tersebut:

```
npm install express --save
```

```
npm install socket.io --save
```

Selanjutnya, buat file baru bernama "bab-8-websocket-express-socketio.js" di folder "bab-8", kemudian isi dengan kode ini (kode ini untuk server) (Menurut Socket.io):

```
var app = require('express')();
var server = require('http').Server(app);
var io = require('socket.io')(server);

server.listen(654);

app.get('/', function (req, res) {
  res.sendFile(__dirname + '/bab-8-websocket-express-socketio.html');
});

io.on('connection', function (socket) {

  socket.on('event_ke_server', function (data) {
    console.log(data);
  });

  socket.emit('salam', {
    content: 'Selamat Coding!'
  });

  socket.emit('salam1', {
    content: 'Selamat Coding 1!'
  });
});
```

Kemudian, buat file baru bernama "bab-8-websocket-express-socketio.html" pada folder yang sama, lalu isi dengan kode ini (kode ini untuk client) (Menurut Socket.io):

```
<!DOCTYPE html>
<html>

<head>
  <style>
    div {
      display: inline;
```

```

    }
  </style>
</head>

<body>
  <h3>Percobaan Express.js dan Socket.io</h3><br>
  Pesan: <div id='pesan'></div><br>
  <script src="/socket.io/socket.io.js"></script>
  <script>
    var socket = io.connect('http://localhost:654');
    socket.on('salam', function (data) {
      document.getElementById('pesan').innerHTML = data.content;
      socket.emit('event_ke_server', {
        name: 'event_ke_server',
        type: 'event'
      });
    });
  </script>
</body>

</html>

```

Sekarang, jalankan script "bab-8-websocket-express-socketio.js" dengan PowerShell/Terminal:

```
node bab-8-websocket-express-socketio.js
```

Kemudian, buka browser ke:

```
http://localhost
```

Nanti, di browser akan muncul output:

```
Pesan: hello world!
```

Sedangkan pada PowerShell/Terminal, akan muncul output:

```
{ name: 'event_ke_server', type: 'event' }
```

Bab 9. Debugging

Pada aplikasi Node.js yang tidak terlalu kompleks, kita bisa menggunakan `console.log()` saja untuk melakukan debugging.

Tapi apabila aplikasi Node.js kita sangat kompleks kita membutuhkan cara untuk melakukan debugging yang lebih lanjut.

Node.js telah menyediakan fitur debuggingnya secara default.

Debugger ini diakses melalui PowerShell/Terminal pada saat menjalankan script Node.js.

Selain itu, jika kita menggunakan Visual Studio Code, kita bisa melakukan debugging dengan menggunakan GUI.

Untuk mencobanya, buatlah project baru dengan NPM bernama "bab-9":

```
npm init
```

Selanjutnya buat file baru di dalam folder "bab-9" yang bernama "bab-9-coba-debug.js", kemudian isi dengan kode ini:

```
var someVar = "xcvxcvxcv";
debugger;

setTimeout(function(){
  someVar = "ABCDE";
  debugger;
}, 1000);

someVar = "123";
debugger;
```

Tampak pada beberapa baris kode di atas ada keyword:

```
debugger;
```

Keyword tersebut adalah breakpoint.

Dengan keyword tersebut, program Node.js bisa dihentikan di tengah-tengah jalannya program dan variabel yang ada pada posisi di atasnya dapat dianalisis.

Debugging dengan PowerShell/Terminal

Untuk men-debug script yang kita buat tadi, buka PowerShell/Terminal, kemudian jalankan perintah:

```
node inspect bab-9-coba-debug.js
```

Nanti, akan muncul output semacam ini:

```
node inspect bab-9-coba-debug.js
< Debugger listening on ws://127.0.0.1:9229/36e0ecf2-3e16-4e11-a991-21148487e2c2
< For help see https://nodejs.org/en/docs/inspector
Break on start in cobadebug.js:1
> 1 (function (exports, require, module, __filename, __dirname) { var someVar =
"xcvxcvxcv";
  2 debugger;
  3
debug>
```

Sampai di sini, di mana ada prompt bernama "debug>", kita bisa melakukan beberapa hal:

- Input 'c' untuk melanjutkan jalannya script sampai menyentuh breakpoint
- Input 'n' untuk melanjutkan ke baris kode selanjutnya
- Input 's' untuk memasuki sebuah fungsi
- Input 'o' untuk menyelesaikan fungsi, kemudian keluar dari fungsi

Sekarang kita coba masukkan 'c':

```
debug> c
```

Nanti akan muncul output:

```
break in bab-9-coba-debug.js:2
  1 (function (exports, require, module, __filename, __dirname) { var someVar =
"xcvxcvxcv";
> 2 debugger;
  3
  4 setTimeout(function(){
debug>
```

Tanda panah (">") pada baris ke-2 menunjukkan bahwa kode kita berhenti di baris ke-2 dari script.

Karena variabel "someVar" ada di atasnya, maka sekarang kita bisa melihat nilainya melalui REPL.

Caranya, ketik repl pada prompt "debug>":

```
debug> repl
```

Sekarang kita sudah masuk ke mode REPL.

Di sini kita bisa melihat nilai variabel "someVar" dengan cara:

```
>someVar
```

Setelah memasukkan perintah tadi, akan muncul nilai someVar:

```
'xcvxcvxcv'
```

Selain fungsi-fungsi tadi, kita juga bisa mengawasi nilai variabel "someVar" sepanjang debugging tanpa harus bolak-balik menggunakan repl.

Caranya yaitu dengan menggunakan fungsi watch.

Untuk membuat watch pada variabel "someVar", kita perlu memberi perintah:

```
watch('someVar')
```

Pada prompt "debug>".

Jadi, jika kita sekarang masih dalam mode REPL, keluar dulu dengan mengetik "ctrl+c" sampai prompt "debug>" muncul kembali.

Setelah prompt "debug>" muncul kembali, sekarang jalankan perintah:

```
debug> watch('someVar')
```

Selanjutnya, jalankan perintah ini untuk melihat seluruh variabel yang diawasi dengan watch:

```
debug>watchers
```

Akan muncul output:

```
0: someVar = 'xcvxcvxcv'
```

Sekarang, kita akan memastikan bahwa apa yang diawasi ini bisa berubah-ubah.

Untuk itu, jalankan perintah 'c' sampai breakpoint selanjutnya disentuh:

```
debug>c
```

Kemudian jalankan perintah 'watchers' lagi:

```
debug>watchers
```

Nanti akan kita lihat hasilnya:

```
0: someVar = '123'
```

Lanjutkan lagi sampai breakpoint selanjutnya, yang ada di setTimeout:

```
debug>c
```

Kemudian lihat watcher lagi:

```
debug>watchers
```

Hasilnya seperti ini:

```
0: someVar = 'ABCDE'
```

Untuk keluar dari debugging, ketik perintah ini:

```
debug> .exit
```

Exit tadi ada tanda titiknya di depan (".exit").

Debugging dengan Visual Studio Code

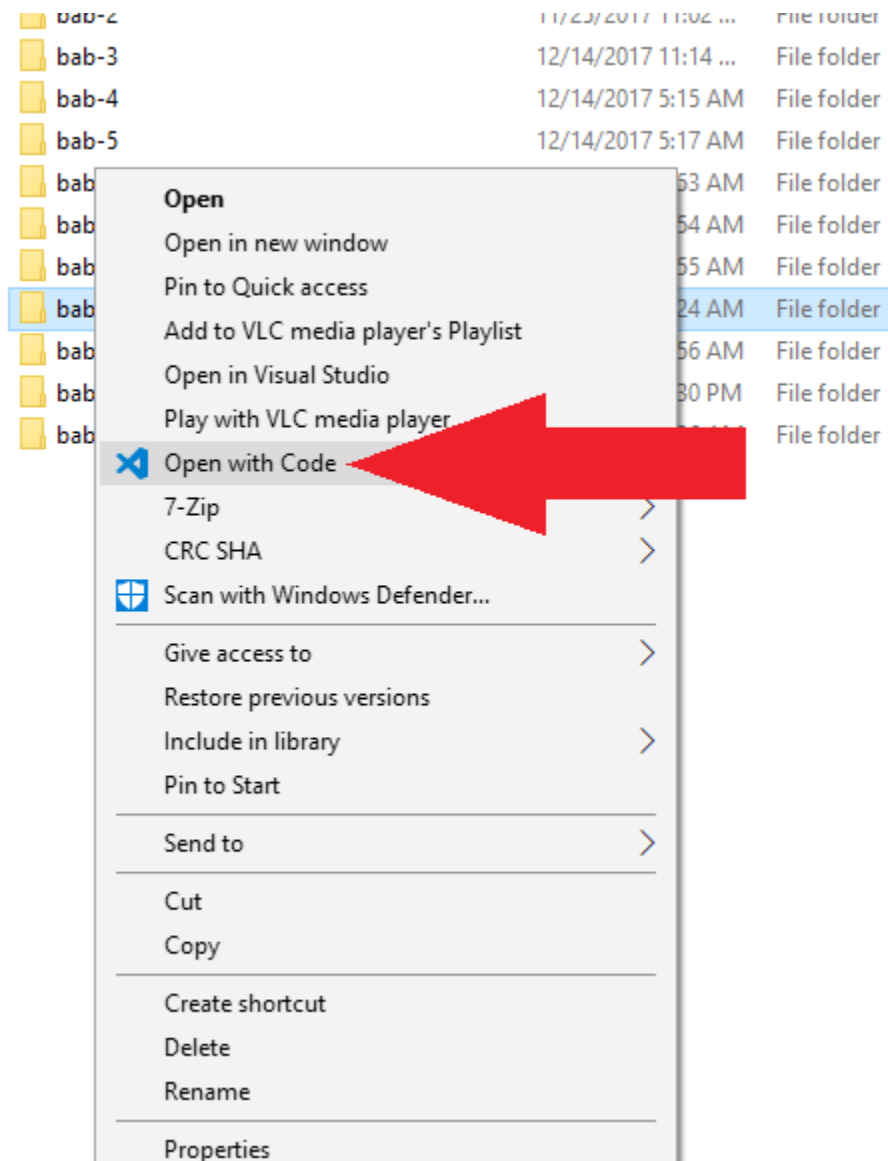
Melakukan debugging dengan Visual Studio Code lebih mudah.

Syaratnya, versi Visual Studio Code yang digunakan adalah 1.10 ke atas

Untuk memulai debugging, buka script "bab-9-coba-debug.js" sebagai keseluruhan folder.

Maksud saya, kita akan membuka script tadi bukan langsung ke file-nya, tapi ke folder yang mengandunginya, yakni folder "bab-9".

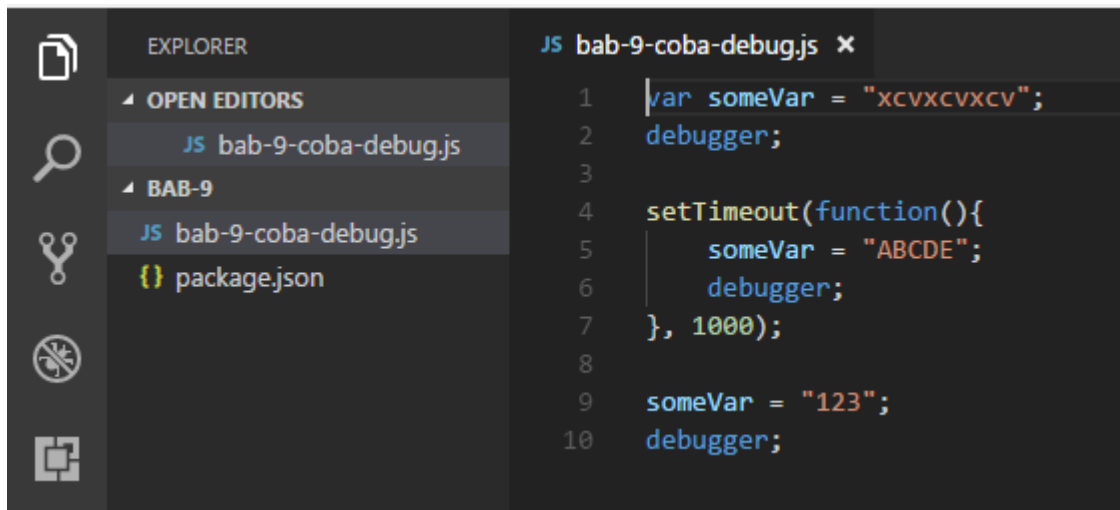
Lebih jelasnya lihat gambar ini:



Buka File Script Sebagai Keseluruhan Folder

Setelah seluruh isi folder "bab-9" beserta package.json-nya terbuka, buka script "bab-9-coba-debug.js" di dalam Visual Studio Code.

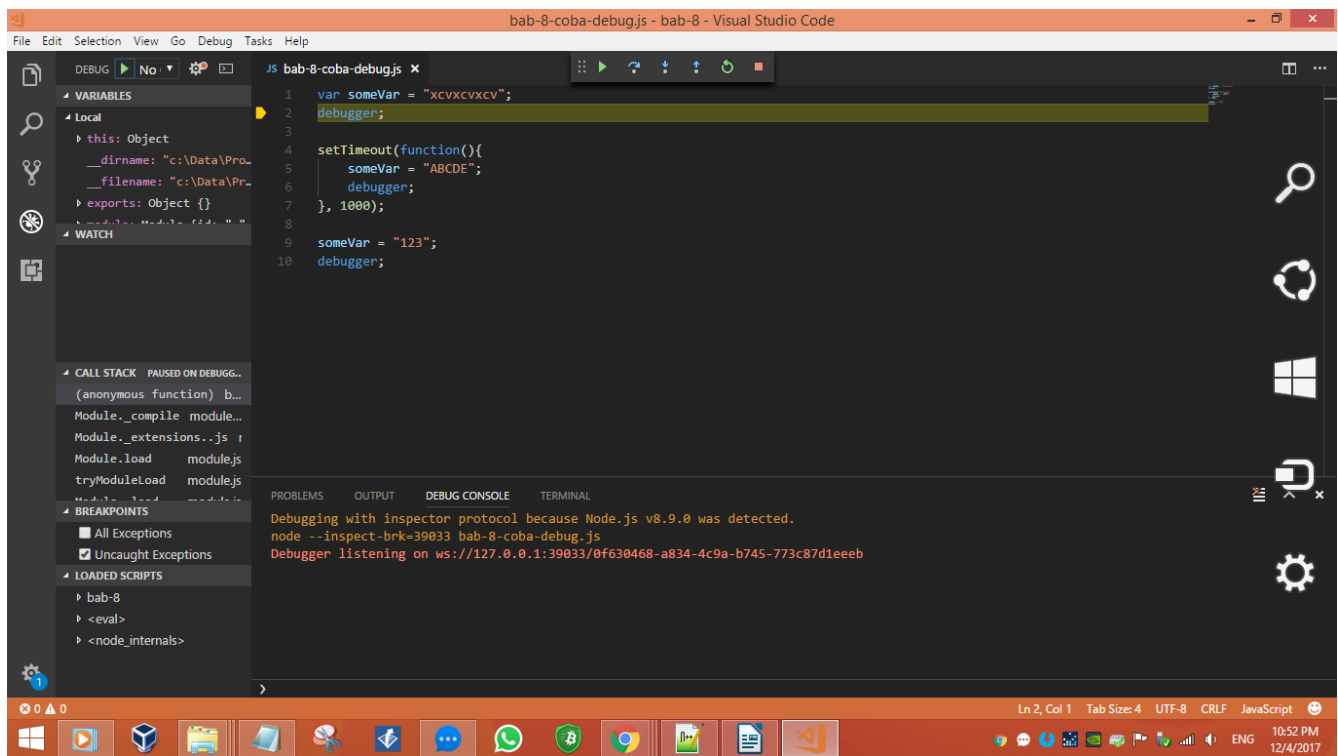
Lihat gambar di bawah ini:



Buka Script di dalam Visual Studio Code

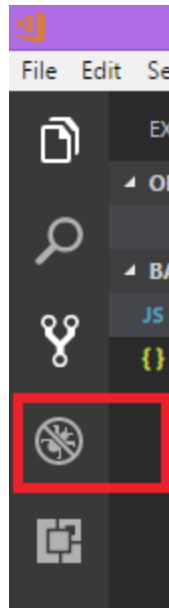
Setelah script terbuka, kita bisa memulai debugging secara visual dengan menekan tombol F5 atau dengan menuju Menu | Debug | Start Debugging.

Tampilan debugger-nya seperti ini:



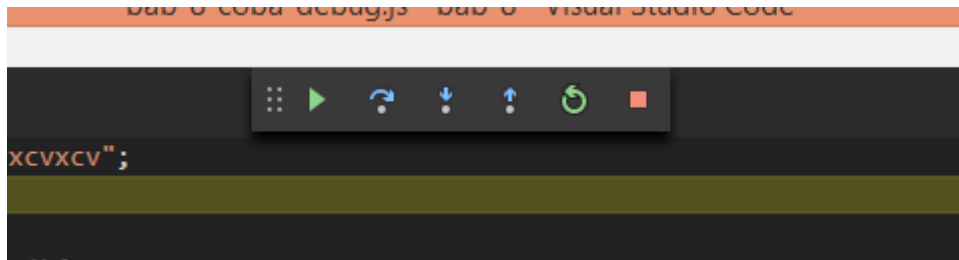
Tampilan Debugger Visual Studio Code

Jika panel debugger di sebelah kiri tidak muncul, tekan icon serangga di sidebar kiri:



*Icon
Serangga*

Dengan Visual Studio Code, kontrol utama untuk debugging seperti 'c', 'n', 's', dan 'o' digantikan oleh tombol yang ada pada tengah layar bagian atas seperti pada gambar di bawah ini:



Tombol Debugging Utama

Untuk mengetahui fungsi dari setiap tombol, arahkan mouse untuk melayang di atas tombol-tombol tersebut.

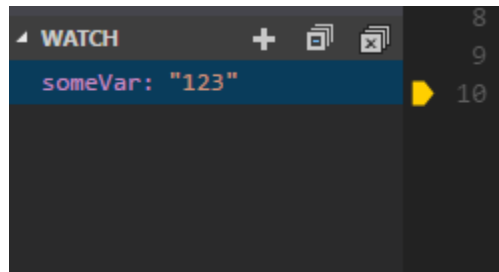
Nanti akan keluar keterangannya.

Daftar variabel local dan global ada di panel kiri yang bernama "VARIABLES".

Nanti variabel "someVar" dalam kode kita akan muncul di bagian ini setiap kali

debugger menyentuh breakpoint.

Untuk membuat watch, bisa dilakukan dengan menekan tombol "+" yang ada pada panel "WATCH" (lihat gambar di bawah ini).



Membuat Watch

Di panel "WATCH" kita bisa memonitor perubahan dari nilai variabel yang diawasi.

Bab 10. Unit Testing

Dalam pemrograman, kita dituntut untuk menulis kode yang berkualitas.

Dengan menulis kode yang berkualitas, diharapkan kita dapat meminimalisasi kesalahan dalam kode tersebut.

Salah satu cara untuk meminimalisasi kesalahan dalam kode adalah dengan melakukan unit testing.

Inti dari unit testing adalah menguji fungsi-fungsi yang telah kita tulis agar sesuai dengan spesifikasi yang kita syaratkan.

Misalnya, jika kita memiliki fungsi `tambah(a, b)` yang mengembalikan nilai penjumlahan antara `a` dan `b`, kita perlu menguji bahwa jika `a = 2` dan `b = 2` hasilnya adalah 4 dan angka 4 ini merupakan variabel bertipe number.

Untuk tujuan itu, NPM memiliki modul `mocha` dan `chai`, sedangkan modul bawaan Node.js sendiri menyediakan `Assert`.

Sekarang kita akan membuat beberapa script sederhana untuk di-test.

Buat folder baru bernama "bab-10", kemudian buat project Node.js bernama "bab-10" dengan `npm init`.

Isi pertanyaan dari `npm init` sedemikian rupa sehingga hasil dari `package.json` seperti ini:

```
"name": "bab-10",
"version": "1.0.0",
"description": "",
"main": "index.js",
"scripts": {
  "test": "mocha"
},
"author": "Anonim",
"license": "UNLICENSED",
```

Selanjutnya, install `mocha` dan `chai`:

```
npm install mocha --save-dev
```



```
npm install chai --save-dev
```

Setelah ini, kita akan membuat dua buah modul buatan sendiri yang sederhana, yakni myModule.js dan myModule1.js

Script "myModule.js" akan di-test dengan modul Assert bawaan Node.js, sedangkan script "myModule1.js" akan di-test dengan modul Assert dari chai.

Walaupun demikian, keduanya akan menggunakan mocha sebagai test framework.

Berikut ini isi dari script "myModule.js":

```
module.exports = {  
  message1: function () {  
    return "message1";  
  },  
  
  kurang: function (a, b) {  
    return a - b;  
  }  
}
```

Sedangkan isi dari script "myModule1.js" seperti ini:

```
module.exports = {  
  message2: function () {  
    return "message2";  
  },  
  
  kali: function (a, b) {  
    return a * b;  
  }  
}
```

Selanjutnya, buatlah folder baru di dalam folder "bab-10" bernama "test".

Di dalam folder tersebut, buat script baru bernama "myModuleTest.js" dan "myModule1Test.js".

Isi dari script "myModuleTest.js" adalah:

```

const assert = require("assert");
const message1 = require("../myModule").message1;
const kurang = require("../myModule").kurang;

describe("myModule", function () {
  it("message1 harus mengembalikan 'message1'", function () {
    assert.equal(message1(), "message1");
  });

  it("kurang harus mengembalikan nilai kurang", function () {
    assert.equal(kurang(5, 2), 3);
  });
});

```

Tampak pada bagian teratas bahwa kita menggunakan modul assert bawaan dari Node.js.

Sedangkan isi dari script "myModule1Test.js" adalah:

```

const assert = require("chai").assert;
const message2 = require("../myModule1").message2;
const kali = require("../myModule1").kali;

describe("myModule1", function () {
  it("message2 harus mengembalikan 'message2'", function () {
    assert.equal(message2(), "message2");
  });

  it("message2 harus mengembalikan tipe data string", function () {
    assert.typeOf(message2(), "string");
  });

  it("kali harus mengembalikan nilai kali", function () {
    assert.equal(kali(10, 2), 20);
  });

  it("kali harus mengembalikan tipe data number", function () {
    assert.typeOf(kali(2, 2), "number");
  });
});

```

Di script ini tampak bahwa kita menggunakan modul Assert dari Chai.

Sekarang, buka PowerShell/Terminal ke folder "bab-10".

Lalu jalankan perintah ini:

```
npm run test
```

Output-nya akan seperti ini:

```
myModule1
  ✓ message2 harus mengembalikan 'message2'
  ✓ message2 harus mengembalikan tipe data string
  ✓ kali harus mengembalikan nilai kali
  ✓ kali harus mengembalikan tipe data number

myModuleAsync
  ✓ asyncSub(20, 11) harus mengembalikan '9'

myModule
  ✓ message1 harus mengembalikan 'message1'
  ✓ kurang harus mengembalikan nilai kurang

7 passing (37ms)
```

Tampak bahwa test berhasil sepenuhnya.

Seandainya kita coba mengubah isi script "myModule1.js" menjadi seperti ini:

```
module.exports = {
  message2: function () {
    return 1;
  },

  kali: function (a, b) {
    return a * b;
  }
}
```

Maka outputnya akan seperti ini:

```
myModule1
  1) message2 harus mengembalikan 'message2'
  2) message2 harus mengembalikan tipe data string
  ✓ kali harus mengembalikan nilai kali
  ✓ kali harus mengembalikan tipe data number

myModuleAsync
```

```
✓ asyncSub(20, 11) harus mengembalikan '9'

myModule
  ✓ message1 harus mengembalikan 'message1'
  ✓ kurang harus mengembalikan nilai kurang

5 passing (24ms)
2 failing

1) myModule1
   message2 harus mengembalikan 'message2':
   AssertionError: expected 1 to equal 'message2'
   at Context.<anonymous> (test\myModule1Test.js:7:16)

2) myModule1
   message2 harus mengembalikan tipe data string:
   AssertionError: expected 1 to be a string
   at Context.<anonymous> (test\myModule1Test.js:11:16)
```

Di sini terlihat bahwa ada 2 kesalahan pada script "myModule1.js", yakni tipe datanya bukan string dan nilai yang dikembalikan bukan "message2".

Perlu diketahui bahwa kita menguji kebenaran nilai yang dikembalikan adalah dengan assert equal:

```
it("message1 harus mengembalikan 'message1'", function () {
  assert.equal(message1(), "message1");
});
```

Hal itu berlaku untuk Assert bawaan Node.js dan Chai:

```
it("message2 harus mengembalikan 'message2'", function () {
  assert.equal(message2(), "message2");
});
```

Tapi dalam hal fitur, Chai memiliki assertion yang tidak dimiliki Assert bawaan Node.js, yaitu:

```
it("message2 harus mengembalikan tipe data string", function () {
  assert.typeOf(message2(), "string");
});
```

Yaitu assert typeOf.

Jadi, lebih baik kita menggunakan Chai karena lebih lengkap fiturnya.

Selain itu, kita juga bisa melakukan unit testing pada fungsi asynchronous.

Untuk mencobanya, buat fungsi asynchronous di dalam script bernama "myModuleAsync.js".

Isinya seperti ini:

```
module.exports.asyncSub = function (a, b, callback) {  
  setTimeout(function () {  
    callback(a - b);  
  }, 1000);  
};
```

Selanjutnya buat script "myModuleAsnycTest.js" di dalam folder "bab-10/test".

Isinya seperti ini:

```
const assert = require("chai").assert;  
const asyncSub = require("../myModuleAsync").asyncSub;  
  
describe("myModuleAsync", function () {  
  it("asyncSub(20, 11) harus mengembalikan '9'", function () {  
    asyncSub(20, 11, function (res) {  
      assert.equal(res, 9);  
      done();  
    });  
  });  
});
```

Lalu jalankan:

```
npm run test
```

Hasilnya:

```
myModule1  
  ✓ message2 harus mengembalikan 'message2'  
  ✓ message2 harus mengembalikan tipe data string  
  ✓ kali harus mengembalikan nilai kali  
  ✓ kali harus mengembalikan tipe data number  
  
myModuleAsync
```

✓ asyncSub(20, 11) harus mengembalikan '9'

myModule

✓ message1 harus mengembalikan 'message1'

✓ kurang harus mengembalikan nilai kurang

7 passing (37ms)

Bab 11. Deployment

Setelah kita menyelesaikan program Node.js, kita mungkin perlu meng-host program kita pada sebuah hosting agar program kita dapat diakses melalui internet.

Saat ini sudah cukup banyak jasa penyedia hosting Node.js di dalam maupun di luar negeri.

Jenis hosting yang bisa digunakan untuk Node.js adalah VPS, Dedicated Server, dan Container Application Platform.

VPS yang tersedia di dalam negeri bisa dicari di google dengan keyword:

VPS yang tersedia di luar negeri bisa dicari di google dengan keyword:

Jika ingin mencari dedicated server, tinggal ganti keyword "VPS" dengan "Dedicated Server".

Sedangkan, untuk jenis Container Application Platform, kita bisa cari dengan keyword:

Atau jika ingin mencari yang lokal:

Cara Men-Deploy Program Node.js di Heroku

Sekarang, kita akan mencoba melakukan deployment program Node.js pada Container Application Platform.

Karena saya ingin yang gratis, saya akan mencobanya di Heroku.

Sebelumnya, kita harus memiliki sebuah aplikasi web terlebih dahulu untuk dicoba.

Jadi, buatlah project baru pada folder "bab-11" dengan NPM, kemudian buat script bernama "index.js".

Berikut ini adalah isi dari file "index.js":

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

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.end('Hello World!');
}).listen((process.env.PORT || 5000)); //YANG INI
```

Perhatikan bagian ini:

```
}).listen((process.env.PORT || 5000));
```

Di situ kita menggunakan process.env.PORT agar port yang ditentukan tergantung dari environment variable.

Di Heroku, kita tidak bisa menentukan di port berapa program Node.js kita berjalan.

Oleh karena itu, kita menggunakan port sesuai dengan yang ada pada environment variable.

Juga, pastikan file "package.json" kita seperti ini:

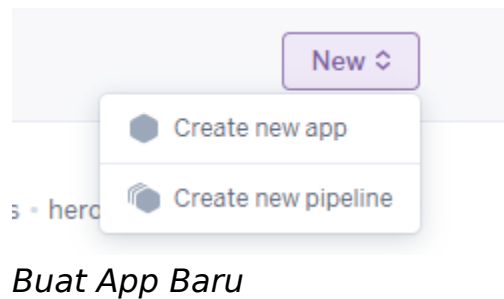
```
{
  "name": "bab-10",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start": "node index.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Anonim",
  "license": "UNLICENSED"
}
```

Pastikan yang digarisbawahi di package.json tadi ada.

Sekarang persiapan kode kita selesai.

Selanjutnya, daftarkan akun Anda terlebih dahulu di Heroku (<https://www.heroku.com/>).

Kemudian, buat app baru dan beri nama "bab-11" jika nama tersebut tersedia:



App name

Choose a region

 United States 

Create app

Isian App Name

Setelah selesai, Anda siap untuk menguploadnya.

Pastikan Anda sudah menginstall semua software yang dibutuhkan:

- Git For Windows(<https://git-for-windows.github.io/>)
- Heroku Client (<https://devcenter.heroku.com/articles/heroku-cli>)

Sekarang buka command line, masuk ke folder "bab-11":

```
cd bab-11
```

Jalankan perintah ini:

```
heroku login
```

Isi email dan password Anda.

Kemudian jalankan perintah ini:

```
git init  
heroku git:remote -a bab-11
```

```
git add .  
git commit -am "make it better"  
git push heroku master
```

Tunggu sampai proses upload selesai.

Setelah itu buka browser ke:

```
https://bab-11.herokuapp.com/
```

Untuk mengupdate saja, cukup dengan perintah ini:

```
heroku login  
git add .  
git commit -am "make it better"  
git push heroku master
```

Nama subdomain sama dengan nama App yang telah kita berikan.

Kalau Anda tidak bisa membuatnya dengan nama "bab-11", tinggal ganti saja perintah-perintah tadi dengan nama project Anda.

Cara Men-Deploy Program Node.js di Ubuntu Server

Untuk men-deploy program Node.js di Ubuntu Server, kita harus memastikan bahwa kita telah meng-install Node.js di Ubuntu Server.

Cara menginstall Node.js di Ubuntu Server telah kita bahas pada Bab 2.

Selanjutnya kita juga perlu meng-install PM2.

PM2 adalah process manager untuk Node.js.

Dengan PM2, kita bisa menjalankan dan menghentikan script Node.js dengan comand line.

Agar bisa meng-install-nya di Ubuntu Server, kita memerlukan aplikasi Putty jika sistem operasi client menggunakan Windows.

Download Putty di sini:

```
http://www.putty.org/
```

Selanjutnya, buka Putty ke IP Address VPS kita.

Setelah login ke SSH, kita bisa menginstall PM2 dengan perintah ini:

```
sudo npm install -g pm2
```

Jika katakanlah, kita memiliki sebuah script bernama "hello.js", maka kita bisa menjalankan script tersebut dengan PM2 seperti ini:

```
pm2 start hello.js
```

Sedangkan, untuk menghentikannya, gunakan perintah ini:

```
pm2 stop hello.js
```

Untuk me-restart-nya, gunakan perintah ini:

```
pm2 restart hello.js
```

Untuk melihat daftar program Node.js yang dijalankan PM2, gunakan perintah ini:

```
pm2 list
```

Untuk melihat status, CPU, dan memory usage, gunakan perintah ini:

```
pm2 monit
```

Cara Men-Deploy Banyak Program Node.js dalam satu Ubuntu Server Menggunakan Apache

Kalau kita perhatikan dari pembahasan sebelumnya, kita melihat bahwa server Node.js hanya berjalan pada satu IP address.

Yang membedakan server Node.js yang satu dengan yang lain adalah nomor port-nya.

Sedangkan, pada praktiknya, ada kemungkinan kita membutuhkan server Node.js agar bisa berjalan pada subdomain yang berbeda tetapi dengan nomor port yang sama.

Untuk melakukan itu, kita memerlukan sebuah proxy agar alamat:

```
http://www.domainsaya.com
```

Dengan alamat:

```
http://subdomain.domainsaya.com
```

Bisa mengakses dua server Node.js yang berbeda.

Proxy semacam ini bisa diperoleh dengan menggunakan Nginx maupun Apache.

Tapi yang akan kita bahas kali ini adalah Apache.

Pertama, pastikan kita telah meng-install Apache pada Ubuntu Server kita.

Jika belum, silakan install dahulu:

```
sudo apt-get install apache2
```

Kemudian buka file konfigurasi apache dengan cara:

```
sudo nano /etc/apache2/apache2.conf
```

Cari bagian yang ini:

```
<VirtualHost *:80>  
...  
</VirtualHost>
```

Kemudian ganti dengan ini:

```

<VirtualHost *:80>
    ServerAdmin admin@domainsaya.com
    ServerName www.domainsaya.com

    ProxyRequests off

    <Proxy *>
        Order deny,allow
        Allow from all
    </Proxy>

    <Location />
        #isi sesuai port number program Node.js kita
        ProxyPass http://localhost:8081/
        ProxyPassReverse http://localhost:8081/
    </Location>
</VirtualHost>
<VirtualHost *:80>
    ServerAdmin admin@domainsaya.com
    ServerName subdomain.domainsaya.com

    ProxyRequests off

    <Proxy *>
        Order deny,allow
        Allow from all
    </Proxy>

    <Location />
        #isi sesuai port number program Node.js kita
        ProxyPass http://localhost:8082/
        ProxyPassReverse http://localhost:8082/
    </Location>
</VirtualHost>

```

Sekarang, <http://www.domainsaya.com> akan diarahkan ke <http://localhost:8081> dan <http://subdomain.domainsaya.com> akan diarahkan ke <http://localhost:8082>

Bab 12. C++ Addons

Kita tahu bahwa npmJS memiliki modul yang sangat banyak.

Walaupun begitu, tidak semua modul yang kita butuhkan selalu ada.

Mungkin saja, salah satu library C++ favorit kita yang kita perlukan fungsinya tidak tersedia di npmJS.

Atau mungkin, kita perlu membuat fungsi yang membutuhkan kerja prosesor yang berat jika menggunakan Javascript, tapi lebih ringan dengan menggunakan C++.

Semua itu ada solusinya.

Node.js menyediakan Native Abstraction untuk menghubungkan script Node.js dengan C++.

Dengan cara ini, kita bisa memanggil fungsi yang ditulis dalam bahasa C++ melalui script Node.js.

Cara ini membutuhkan package NPM yang bernama NAN (Native Abstractions for Node.js).

Walaupun begitu, NAN tidak berjalan sendiri.

Dibutuhkan package node-gyp dan compiler yang berbeda-beda pada sistem operasi yang berbeda.

Sekarang, kita akan mencoba membuat aplikasi hello world dalam bahasa C++ yang akan dipanggil melalui script Node.js.

Untuk mencobanya, kita perlu menginstall beberapa software tambahan.

Untuk Windows, ini yang kita perlukan:

- Microsoft Visual Studio Community 2017
- Python 2.7
- node-gyp

Sedangkan untuk Ubuntu, ini yang kita perlukan:

- g++
- make
- Python 2.7
- node-gyp

Keduanya membutuhkan node-gyp dan Python 2.7, karena itu adalah build tool-nya, tapi compiler dari masing-masing sistem operasi berbeda.

Membuat Program Hello World C++ di Windows

Persiapan

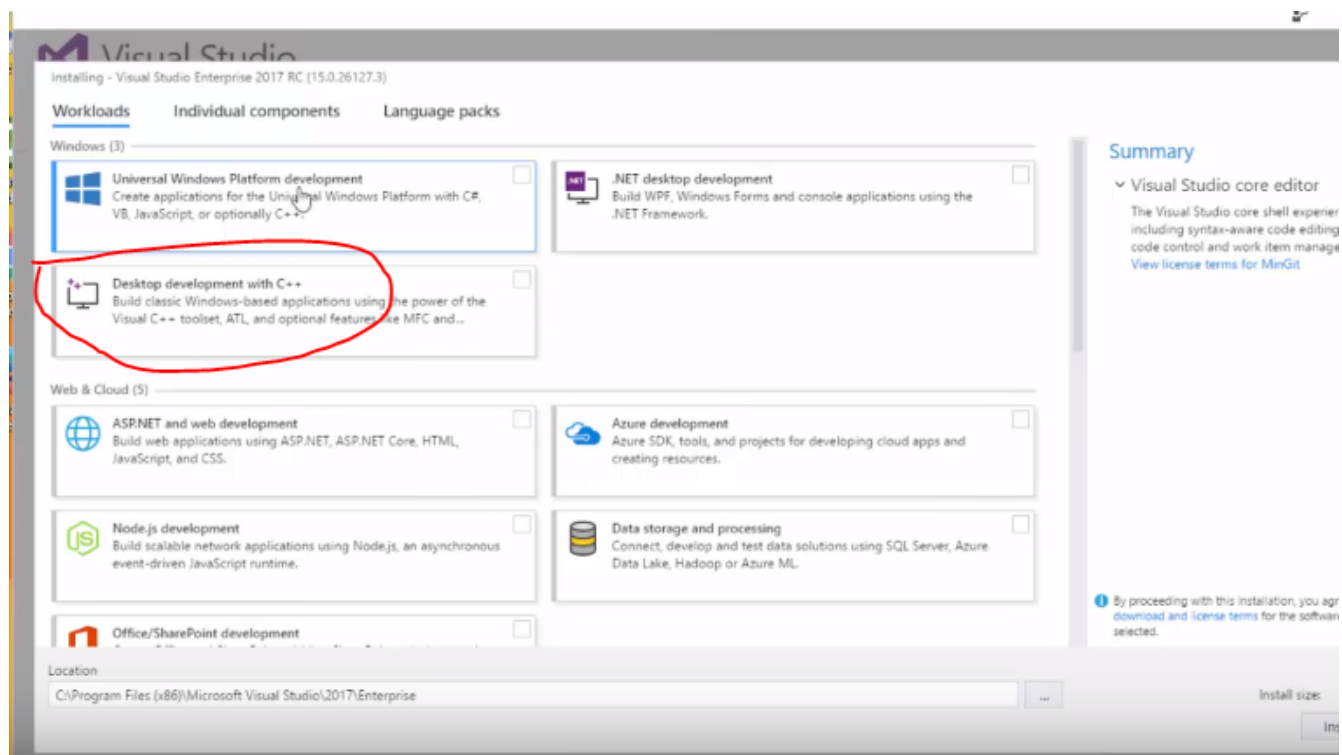
Sekarang, kita harus menginstall Visual Studio Community 2017 secara sukses dan tanpa kegagalan sedikitpun.

Oleh karena itu, kita harus yakin bahwa sistem operasi Windows yang kita gunakan sudah ter-update.

Untuk men-download Visual Studio Community 2017, kunjungi:

<https://www.visualstudio.com/downloads/>

Selanjutnya install dan pastikan kita memilih "Desktop Development with C++" seperti pada gambar ini:



Selanjutnya, download Python 2.7 dari alamat ini:

<https://www.python.org/download/releases/2.7/>

Kemudian install secara default.

Langkah selanjutnya adalah menginstall node-gyp.

Untuk meng-install-nya, jalankan perintah ini:

```
npm install -g node-gyp
```

Setelah semua software yang diperlukan terinstall, pastikan versi Visual Studio Community 2017 compiler (build tools) dan path Python 2.7 sudah di-set dengan benar.

Caranya, jalankan perintah-perintah ini:

```
npm config set msvs_version 2017  
npm config set python path/ke/lokasi/python 2.7
```

Menulis Kode Program Hello World

Selanjutnya, buat folder baru bernama "bab-12" dan di dalamnya, buat folder baru lagi bernama "bab-12-hello-world-c++-windows".

Di dalam folder "bab-12-hello-world-c++-windows", buat sebuah file C++ bernama "main.cpp", yang isinya (menurut NAN):

```
#include <nan.h>  
  
NAN_METHOD(Hello) {  
    auto message = Nan::New("Hello dari C++!").ToLocalChecked();  
    info.GetReturnValue().Set(message);  
}  
  
NAN_MODULE_INIT(Initialize) {  
    NAN_EXPORT(target, Hello);  
}  
  
NODE_MODULE(addon, Initialize);
```

Kode itu akan mem-print "Hello dari C++!" dan ditulis dalam bahasa pemrograman C++.

Oleh karena itulah kita perlu menginstall Visual Studio sebelumnya.

Kemudian, buat sebuah script Node.js bernama "main.js", lalu isi dengan kode ini (menurut NAN):

```
const {Hello} = require('./build/Release/addon');  
  
console.log(Hello());
```

Script itu dibuat untuk menampilkan return value dari fungsi "Hello()" yang ditulis dalam C++.

Setelah itu, buat file package.json (tidak usah dengan npm init) berisi:

```
{  
  "name": "bab-12",  
  "version": "1.0.0",  
  "description": "",  
  "main": "main.js",  
  "scripts": {  
    "build": "node-gyp build",  
    "install": "node-gyp rebuild",  
    "start": "node main.js",  
    "test": "echo \"Error: no test specified\" && exit 1"  
  },  
  "author": "Anonim",  
  "license": "UNLICENSED"  
}
```

Melalui PowerShell, selanjutnya kita menginstall modul NAN:

```
npm install nan --save
```

Dan yang terakhir, buat file bernama "binding.gyp" yang isinya (menurut NAN):

```
{  
  "targets": [  
    {  
      "include_dirs": [  
        "<!(node -e \"require('nan')\")"  
      ],  
    },  
  ],  
}
```

```
        "target_name": "addon",
        "sources": [ "main.cpp" ]
    }
]
```

Pada bagian ini:

```
"include_dirs": [
    "<!(node -e \"require('nan')\")"
],
```

Sebenarnya kita menjalankan eval berupa "require('nan')" yang merupakan modul Node.js.

Bagian "sources" adalah file source code C++ yang akan di-build.

Perhatikan juga package.json bagian ini:

```
"build": "node-gyp build",
"install": "node-gyp rebuild",
"start": "node main.js",
```

Dengan menuliskan "install": "node-gyp rebuild", di package.json, NPM akan melakukan rebuild ketika kita memberi perintah:

```
npm install
```

Di dalam folder "bab-12-hello-world-c++-windows".

Sedangkan dengan menuliskan "build": "node-gyp build", kita hanya melakukan build saja.

Adapun "start": "node main.js" berarti menjalankan script main.js, sama halnya seperti melakukan:

```
node main.js
```

Sekarang coba jalankan perintah ini dengan PowerShell **SEBAGAI ADMINISTRATOR**:

```
npm install
```

```
npm start
```

Nanti output-nya akan seperti ini:

```
> bab-11@1.0.0 start C:\Data\Projects\NodeJS\Panduan Node.js 8.x dalam 14 Bab\bab-11\bab-11-hello-world-c++-windows
> node main.js

Hello dari C++!
```

Membuat Program Hello World C++ di Ubuntu

Persiapan

Sebelum kita mencoba membuat program ini, kita perlu menginstall Python 2.7 dan dependencies-nya di Ubuntu.

Ubuntu yang kita gunakan adalah Ubuntu Server versi 17.04 64 bit.

Jika Ubuntu Server yang kita gunakan ada di VPS atau Dedicated Server, kita membutuhkan Putty.

Tapi jika Ubuntu Server yang kita gunakan ada di Virtual Machine (seperti VMWare atau VirtualBox) atau ada di PC kita, maka kita hanya membutuhkan Terminal bawaannya.

Untuk meng-install Python 2.7, mulai dari perintah ini di Terminal atau Putty:

```
sudo apt-get install build-essential checkinstall
```

Selanjutnya, install dependencies-nya:

```
sudo apt-get install libreadline-gplv2-dev libncursesw5-dev libssl-dev libsqlite3-dev tk-dev libgdbm-dev libc6-dev libbz2-dev
```

Buatlah folder bernama "Downloads" pada direktori home Anda:

```
mkdir Downloads
```

Kemudian masuk ke dalamnya dan buat variabel berisi "2.7.13". Itu adalah versi Python yang akan kita install:

```
version=2.7.13
```

```
cd ~/Downloads/
```

Selanjutnya, download package Python 2.7.13:

```
wget https://www.python.org/ftp/python/$version/Python-$version.tgz
```

Ekstrak dan masuk ke dalam direktorinya:

```
tar -xvf Python-$version.tgz  
cd Python-$version
```

Selanjutnya, build dan install Python 2.7.13:

```
./configure  
make  
sudo checkinstall
```

Pada perintah-perintah ini, akan muncul pertanyaan. Jawab saja semuanya sesuai yang ditawarkan.

Kalau ditawarkan dengan y, tinggal enter, sedangkan kalau ditawarkan dengan n, tinggal enter saja.

Setelah proses ini selesai, kita akan mendapati Python 2.7.13, g++, dan make terinstall pada Ubuntu Server.

Jangan lupa juga untuk meng-install node gyp:

```
npm install -g node-gyp
```

Menulis Kode Program Hello World

Tahap ini sama dengan Windows.

Tapi, karena kita harus menjalankan npm install dalam mode Administrator, kita harus menggunakan sudo:

```
sudo npm install --unsafe-perm
```

Parameter --unsafe-perm disertakan karena alasan ini (diambil dari [dokumentasi npmJS](#)):

If npm was invoked with root privileges, then it will change the uid to the user account or uid specified by the user config, which defaults to nobody. Set the unsafe-perm flag to run scripts with root privileges.

Setelah kodenya siap, coba jalankan perintah:

```
sudo npm install --unsafe-perm  
npm start
```

Output-nya kurang lebih mirip dengan di Windows:

```
Hello from C++!
```

Bab 13. Praktikum: Membuat Website Clone Pastebin

Setelah kita memahami dasar-dasar dalam pemrograman Node.js, ada baiknya kita segera mempraktikkan ilmu yang telah kita pelajari.

Kali ini kita akan belajar membuat sebuah clone website yang bernama Pastebin:

<https://pastebin.com/>

Salah satu hal yang paling sering dilakukan oleh pengguna komputer adalah copy dan paste.

Bagaimana tidak, hampir setiap hari ada saja sesuatu yang di-copy-paste, entah itu teks, file, gambar, video, dan lain-lain.

Tapi copy dan paste tidak lengkap tanpa adanya kemampuan berbagi atau sharing.

Untuk memberi fitur tersebut, Pastebin menyediakan layanan untuk menyimpan dan berbagi paste.

User yang terdaftar maupun tidak bisa mengakses fitur paste yang disediakan oleh website tersebut.

Setelah user mem-paste apa yang telah di-copy-nya, user diberi URL paste miliknya yang kemudian dapat disebarluaskan.

Walaupun ide tersebut sederhana, kita bisa melihat betapa besar fungsi dari Pastebin.

Sayangnya, website tersebut tidak mengizinkan semua konten paste.

Ada yang tidak boleh dipaste di website tersebut, seperti:

- Daftar email
- Detail login
- Source code curian
- Data hasil hack
- Data/informasi yang dilindungi hak cipta
- Daftar password
- Data perbankan
- Data personal

- Data pornografi
- Link spam

Seperti yang dijelaskan pada <https://pastebin.com/faq#5> :

What is your Acceptable Use Policy?

Broadly speaking, the site was created to help programmers. You are however welcome to post any type of text to Pastebin. Please do not post email lists, password lists or personal information. The "report abuse" feature can be used to flag such pastes and they will be deleted. Do not aggressively spider the site. If you do want to scrape our website, use our [scraping API](#).



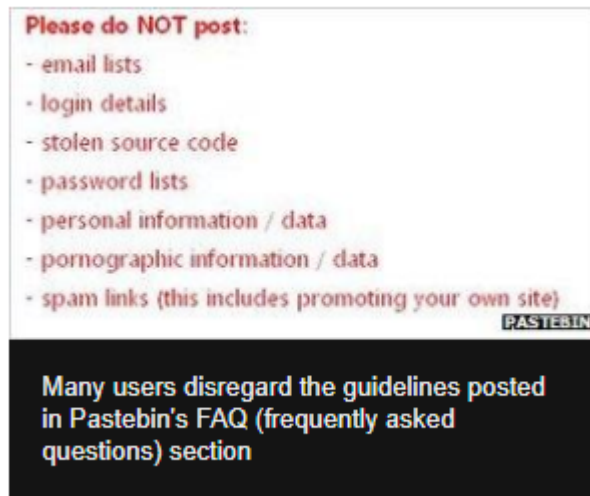
Do NOT post:

- email lists
- login details
- stolen source code
- hacked data
- copyrighted information / data
- password lists
- banking / creditcard / financial information / data
- personal information / data
- pornographic information / data
- spam links (this includes promoting your own site)

If you do not comply with our Acceptable Use Policy we might ban your account and/or IP address from the website. Also, your IP address & user data might be shared with authorities.

Pastebin FAQ

Wajarlah kalau pengguna Pastebin justru cenderung melanggar kebijakan tersebut seperti yang dilansir dari bbc (<http://www.bbc.com/news/technology-17524822>):



Berita tentang Pastebin

Walaupun ide Pastebin sederhana, unique visitor Pastebin pada tahun 2012 mencapai 17 juta.

Traffic yang sangat menggiurkan bukan?

Coba, bayangkan jika kita bisa membuat website seperti itu, lalu ambil potongan kuenya 1 persen saja...

Berarti sudah dapat 170 ribu unique visitor.

Apalagi kalau kita sebagai Admin mengizinkan semua konten dan tidak memerlukan login.

Oleh karena itulah, saya kali ini akan membahas cara membuat clone dari situs Pastebin dengan Node.js agar pembaca dapat membuat Pastebin-nya sendiri dengan kebijakan konten sesuka pembaca.

Aplikasi yang akan kita buat ini bernama "Node.js Pasta".

Memang tidak 100 persen sama dengan Pastebin, tapi setidaknya fungsi utamanya sama.

Harapan saya, setidaknya pembaca dapat 1 persen kue unique visitor-nya.

Yang Anda Butuhkan sebelum Kita Belajar

Sebelum kita mulai, pastikan Anda telah memiliki akun di Heroku.

Heroku digunakan untuk meng-host aplikasi Node.js kita secara gratis, jadi daftarkan diri Anda sebagai free account.

Aplikasi ini memang bisa berjalan di localhost, tapi untuk menge-test tampilan sharing di social media, perlu hosting Node.js di Heroku.

Apabila Anda sudah memiliki hosting lain atau mungkin VPS, silakan digunakan, tapi caranya tidak akan dibahas di sini.

Beberapa software yang harus Anda install terlebih dahulu adalah:

- Node.js (<https://nodejs.org/en/download/>)
- Visual Studio Code (<https://code.visualstudio.com/Download>)
- Git For Windows(<https://git-for-windows.github.io/>)
- Heroku Client (<https://devcenter.heroku.com/articles/heroku-cli>)

Setelah semua software tersebut ter-install, lakukan pengecekan terhadap software-software tersebut.

Pertama, jalankan perintah ini di command line:

```
node -v
```

Hasil yang diharapkan adalah:

```
v8.9.2
```

Kemudian jalankan perintah ini di command line:

```
git --version
```

Hasil yang diharapkan adalah:

```
git version 2.10.2.windows.1
```

Kemudian jalankan perintah ini di command line:

```
heroku -v
```

Hasil yang diharapkan adalah:

```
heroku-cli/6.14.13-6ab5022 (windows-x64) node-v8.4.0
```

Hasil-hasil tersebut tidak harus sama persis karena tergantung versi berapa yang ter-install.

Membedah Aplikasi Pastebin

Halaman Utama (New Paste)

Pastebin memiliki dua fungsi utama: menyimpan teks yang di-paste dan memberi link untuk membagikannya di tempat lain.

Halaman Utama Pastebin

Yang dilakukan oleh pengguna pastebin untuk menyimpan paste adalah dengan membuka halaman berandanya, kemudian mengisi textarea "New Paste".

Selanjutnya pada bagian "Optional Paste Settings", pengguna dapat memilih syntax highlighting dari berbagai macam alternatif yang nanti akan ditampilkan pada halaman paste.

Pengguna juga dapat memilih "Paste Expiration" untuk menentukan usia paste yang dibuat.

Pilihan "Paste Expiration" yang disediakan adalah:

- Never
- 10 Minutes

- 1 Hour
- 1 Day
- 1 Week
- 2 Weeks
- 1 Month
- 6 Month
- 1 Year

Selain itu, pada bagian "Paste Exposure", pengguna dapat memilih:

- Public
- Unlisted

Public berarti paste akan tampil dalam listing "Public Pastes" di bagian kanan dan di halaman "Trends".

Unlisted berarti paste tidak akan tampil di bagian manapun dan hanya bisa dilihat oleh pengguna yang mengetahui link dari paste tersebut.

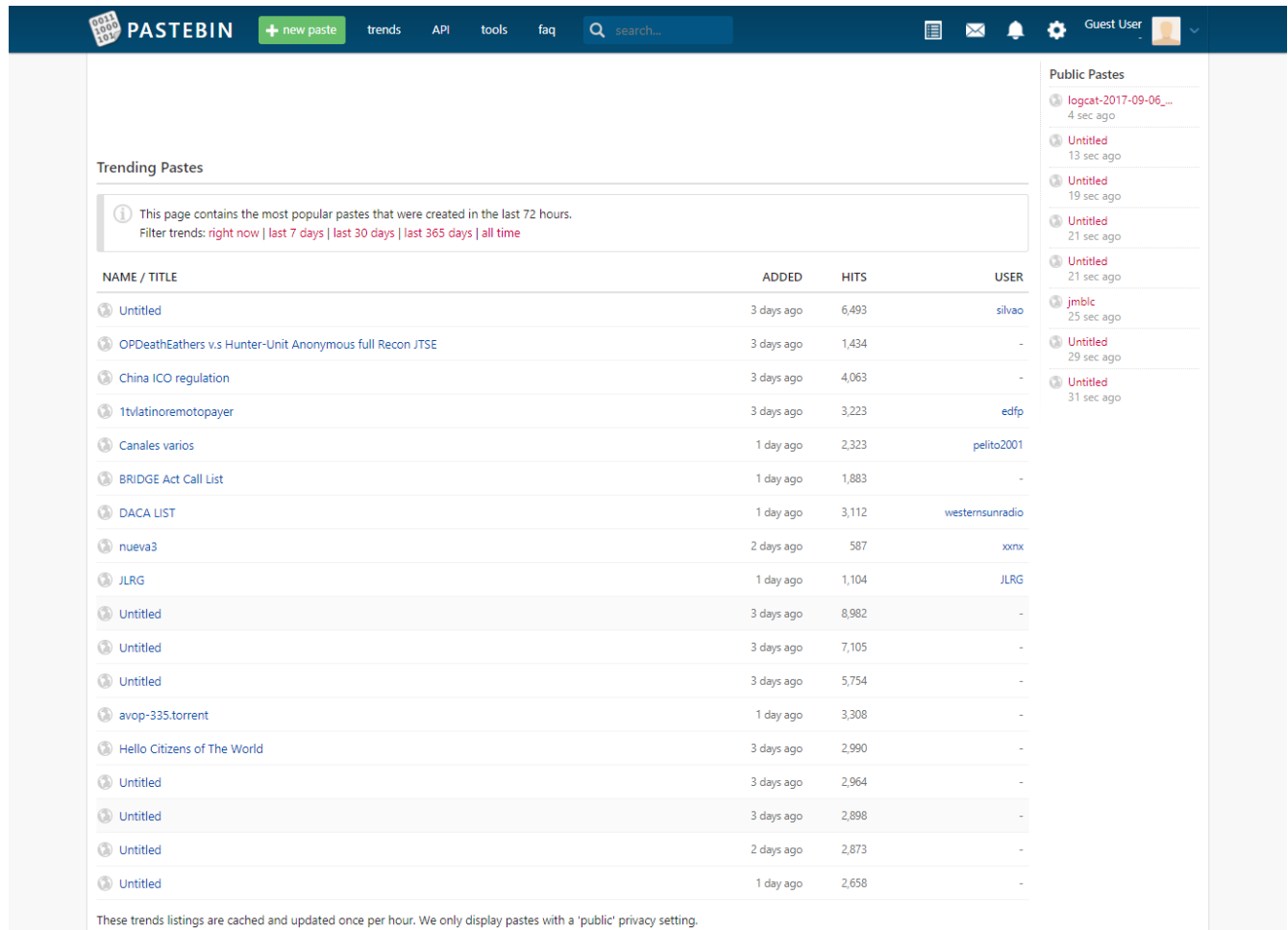
Selain itu, jika pengguna adalah pro member, mereka dapat menempatkan paste pada suatu folder dengan mengisi input "Folder".

Dan option selanjutnya adalah "Title". Di sini pengguna dapat memberi nama paste yang dia buat.

Untuk membedakan member dan mengidentifikasi pro member, Pastebin juga menyediakan form login di sebelah kanan dari "Optional Paste Settings".

Halaman Trends

Pada halaman "Trends", Pastebin menampilkan trending pastes yang ditampilkan dalam bentuk tabel.



PASTEBIN + new paste trends API tools faq search...

Trending Pastes

This page contains the most popular pastes that were created in the last 72 hours.
Filter trends: [right now](#) | [last 7 days](#) | [last 30 days](#) | [last 365 days](#) | [all time](#)

NAME / TITLE	ADDED	HITS	USER
Untitled	3 days ago	6,493	silva0
OPDeathEathers v.s Hunter-Unit Anonymous full Recon JTSE	3 days ago	1,434	-
China ICO regulation	3 days ago	4,063	-
1tvlatinoremotopayer	3 days ago	3,223	edfp
Canales varios	1 day ago	2,323	pelito2001
BRIDGE Act Call List	1 day ago	1,883	-
DACA LIST	1 day ago	3,112	westernsunradio
nueva3	2 days ago	587	xxrx
JLRG	1 day ago	1,104	JLRG
Untitled	3 days ago	8,982	-
Untitled	3 days ago	7,105	-
Untitled	3 days ago	5,754	-
avop-335.torrent	1 day ago	3,308	-
Hello Citizens of The World	3 days ago	2,990	-
Untitled	3 days ago	2,964	-
Untitled	3 days ago	2,898	-
Untitled	2 days ago	2,873	-
Untitled	1 day ago	2,658	-

These trends listings are cached and updated once per hour. We only display pastes with a 'public' privacy setting.

Public Pastes

- logcat-2017-09-06_... 4 sec ago
- Untitled 13 sec ago
- Untitled 19 sec ago
- Untitled 21 sec ago
- Untitled 21 sec ago
- jmbic 23 sec ago
- Untitled 29 sec ago
- Untitled 31 sec ago

Pastebin Trends

Kolom pertama adalah "Name/Title" atau judul dari paste.

Kolom kedua adalah "Added" atau kapan paste tersebut dibuat.

Kolom ketiga adalah "Hits" yang sepertinya adalah jumlah klik dari paste tersebut.

Kolom keempat adalah "User" yakni siapa yang membuat paste tersebut.

Apabila pembuat paste adalah guest atau tidak login terlebih dahulu, maka nama pembuat paste tersebut tidak ditampilkan pada kolom ini.

Halaman Search

Halaman "Search" menampilkan paste yang dicari dengan keyword tertentu.

Tidak ada yang istimewa dari halaman ini, karena halaman ini melakukan search dengan menggunakan google custom search.

Memang cara ini sangat mudah dilakukan dan hemat resource, tapi jika halaman paste tertentu belum di-index google atau jika website ini mengalami deindex, maka hasil pencarian untuk paste tersebut tidak akan muncul.

The screenshot shows the Pastebin website interface. At the top, there's a navigation bar with the Pastebin logo, a '+ new paste' button, and links for 'trends', 'API', 'tools', 'faq', and a search bar. The search bar contains the text 'bakso'. Below the navigation bar, the search results are displayed. The title is 'Search results for: bakso'. It indicates 'About 19 results (0.18 seconds)' and shows a 'Sort by: Relevance' dropdown. The results are powered by Google Custom Search. The first result is a code snippet for a Java program that searches for 'bakso' in a list of items. Other results include links to various websites and documents related to 'bakso', such as 'kw - Pastebin.com', 'alamat lapangan badminton - Pastebin.com', and 'Mama Lover (2012).rar'. On the right side, there's a 'Public Pastes' sidebar showing a list of recent public pastes, including 'Untitled', 'test', 'Recruitment', and 'Hé oé Morfos <3'. At the bottom of the search results, there are two numbered buttons: '1' and '2'.

Pastebin Search

API, Tools, FAQ dan Sisanya

Pastebin juga menyediakan API untuk programmer.

Jadi programmer bisa membuat paste atau mendapat list dari paste atau untuk keperluan lain.

Sedangkan, Tools merupakan daftar aplikasi yang menggunakan API dari Pastebin. Jika kita membuatnya, kita bisa kontak Pastebin agar aplikasi kita di-listing di halaman tersebut.

FAQ cukup jelas.

Sisanya, merupakan fitur untuk pengguna yang sudah login, jadi tidak perlu dibahas.

Merancang Node.js Pasta

Setelah kita membedah Pastebin yang asli, kita bisa mengetahui beberapa fitur utama pada Pastebin.

Beberapa fitur seperti login tidak diperlukan karena kita akan membuat versi lain yang tidak membutuhkan login.

API tidak akan disertakan di sini karena tidak terlalu berguna. Mungkin kita akan membuatnya jika pengguna clone ini sudah semakin banyak.

Jadi, fitur-fitur yang akan disediakan oleh Node.js Pasta adalah:

- Copy-paste teks.
- Syntax highlighting. Dalam kasus ini hanya 3 jenis: Text, HTML, CSS, dan Javascript.
- Paste Expiration. Dalam kasus ini, kita mengubah durasinya menjadi: 1 Day, 12 Hours, 3 Hours, 30 Minutes, dan 2 Minutes.
- Paste Exposure. Dalam kasus ini pilihannya: Public dan Unlisted.
- Paste Title.
- Recent Pastes. Yakni daftar paste yang baru dibuat.
- Paste Manager tanpa login.
- Edit dan Delete Paste dari Paste Manager.
- Paste Manager link dan Paste link generator.
- Trending Pastes.
- Halaman Paste.
- Halaman 404 Not Found. Berguna jika paste yang dibuat sudah expired atau jika pengguna salah memasukkan URL.
- Paste sharing ke social media: Facebook, Twitter, dan Google Plus.

Rancangan User Interface

Node.js Pasta memiliki desain user interface yang sederhana.

Terdiri atas empat halaman, yakni "Home/New Paste", "Paste Manager", "Paste", dan "Trends".

Tidak ada CSS yang rumit karena kita akan menggunakan bootstrap.

Node.js Pasta

Trends [New Paste](#)

Apa itu Node.js Pasta?

Node.js Pasta adalah alat untuk berbagi teks yang di-paste-kan ke website ini.

Anda bisa membuat, mengubah, dan menghapus paste Anda TANPA LOGIN di sini!

Jika berminat dengan website semacam ini, [pekerjaan saya di sini](#) atau [beli bukunya di sini](#).

New Paste

Node.js Pasta

Trends [New Paste](#)

This Paste

```
/*
trik-10.js
Membuat CSV Report tentang Ranking Website di Google
*/
var Nightmare = require('nightmare');
var nightmare = Nightmare({
  show: true
});
var fs = require('fs');
var csv = require('fast-csv');
```

Settings

Syntax Highlighting:

text

Paste Expiration:

1 Day

Paste Exposure:

Public

Title:

scraper-1.js

Save Paste

Delete Paste

Links for This Paste

Manager URL:

http://localhost:5000/manage/utudybd

Copy!

Paste URL:

http://localhost:5000/paste/tcynioc

Copy!

Copyright © Luslikar Sheba 2017

Node.js Pasta

TrendsNew Paste

This Paste

```
1 |
2 /*
3 trik-10.js
4 Membuat CSV Report tentang Ranking Website di Google
5 */
6 var Nightmare = require('nightmare');
7 var nightmare = Nightmare({
8   show: true
9 });
10 var fs = require('fs');
11 var csv = require('fast-csv');
12
13 //-----
14
15 var targetdomain = "www.rakifsi.com";
16 var keywords = [
17   "belajar node js",
18   "membuat aplikasi",
19   "membuat web",
20   "tutorial web scraping"
21 ];
22
```

Links for This Paste

Paste URL:

Copy!

Share



Copyright © Lusfikar Sheba 2017

Halaman "Paste"

Node.js Pasta

TrendsNew Paste

Trending Pastes

scraper-0.js

Expired in 1 Day, 0 Hit(s)

scraper-1.js

Expired in 1 Day, 0 Hit(s)

/* trik-10.js Membuat CSV Report tentang Ranking Website di Google */ var Nightmare = require("n

Copyright © Lusfikar Sheba 2017

Halaman "Trends"

Package NPM yang Dibutuhkan

Untuk mengerjakan aplikasi Node.js Pasta, diperlukan beberapa modul NPM ini.

Body Parser

```
install: npm install body-parser --save
```

Module ini berfungsi untuk memarsing request body.

Expressjs

```
install: npm install express --save
```

Module ini berfungsi sebagai framework untuk routing dan sebagai wrapper untuk menjalankan fungsi Node.js sebagai web server.

EJS

```
install: npm install ejs --save
```

Module ini berfungsi sebagai template engine untuk Expressjs. Dengan module ini kita bisa menyajikan html sebagai template dan memanipulasi nilai tertentu di dalamnya secara dinamis.

Randomstring

```
install: npm install randomstring --save
```

Module ini berfungsi untuk men-generate string acak dengan parameter-parameter tertentu.

SQLite3

```
install: npm install sqlite3 --save
```

Module ini digunakan sebagai DBMS. Keuntungan menggunakan module ini adalah, kita tidak memerlukan server khusus untuk database karena hanya berupa file. Apabila Anda ingin menggantinya menjadi MySQL silakan dicoba sendiri karena tidak dibahas di sini.

Package Non-NPM yang Dibutuhkan

Selain package NPM, dibutuhkan juga package non NPM.

Sebenarnya package-package ini juga tersedia di npmjs.com, tetapi menurut saya lebih mudah menggunakannya tanpa NPM.

jQuery

Package ini adalah library javascript front-end yang berguna untuk memanipulasi HTML dan event-event-nya. Selain itu, Bootstrap juga memerlukan library ini.

Untuk mendapatkannya, download versi terbarunya di:

<https://jquery.com/download/>

Bootstrap

Package ini adalah library javascript dan CSS yang menyediakan berbagai elemen user interface yang cantik. Semua elemen tersebut didesain agar bersifat responsive.

Untuk mendapatkannya, download versi 3-nya di:

<https://getbootstrap.com/docs/3.3/getting-started/#download>

Ace

Package ini adalah library javascript untuk membuat web based code editor.

Sebenarnya kita tidak membutuhkan fungsinya sebagai code editor, tetapi hanya mengambil fungsi syntax highlighting-nya saja.

Bahasa pemrograman yang didukung syntax highlighting-nya sangat banyak, namun kita hanya akan mengambil empat saja: Text, HTML, CSS, dan Javascript.

Untuk mendapatkannya, download versi terbarunya di:

<https://github.com/ajaxorg/ace-builds>

Font Awesome

Font Awesome adalah library yang berisi icon-icon yang sangat banyak. Icon-icon ini adalah font yang berupa vector dan scalable, sehingga tidak akan pecah gambarnya walaupun dalam ukuran berapapun.

Untuk mendapatkannya, download versi terbarunya di:

<http://fontawesome.io/>

Memprogram Node.js Pasta

Sekarang kita telah mengetahui rancangan aplikasi Node.js Pasta dan telah mendapatkan package-package yang dibutuhkan.

Saatnya mulai memprogram.

Tapi sebelumnya, kita buat dahulu struktur direktori project ini agar lebih rapi.

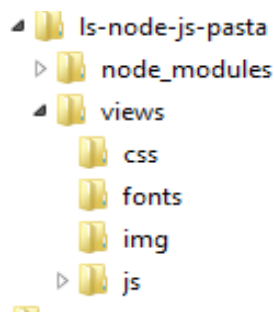
Dan ada satu hal lagi yang perlu saya sampaikan.

Jika Anda ragu apakah kode yang Anda tulis dan penempatannya sudah benar, silakan cek dan samakan dengan kode yang ada pada source code buku ini.

Struktur Direktori Project

Pertama-tama, buatlah folder bernama "ls-node-js-pasta".

Selanjutnya buatlah folder-folder lain di dalamnya sedemikian rupa sehingga strukturnya seperti gambar di bawah ini.



*Struktur Direktori
Node.js Pasta*

Anda boleh melewati folder "node_modules" karena ini akan dibuat secara otomatis ketika Anda menginstall module melalui NPM.

Jika Anda tidak yakin dengan struktur direktori yang Anda buat, silakan cek source code buku ini (di dalam folder "Is-node-js-pasta").

Menempatkan Package-Package Non NPM di Direktori Project

Selanjutnya tempatkanlah package-package non NPM yang telah Anda download ke dalam direktori project sesuai dengan jenis filenya.

File Javascript (extension: .js) dimasukkan ke folder "js".

File Gambar dimasukkan ke folder "img".

File Font (extension: .otf, .eot, .svg, .ttf, .woff, .woff2) dimasukkan ke folder "fonts".

File CSS (extension: .css) dimasukkan ke folder "css".

Khusus untuk gambar, silakan copy dari source code buku ini, karena file gambar tersebut tidak ada di internet.

Anda boleh menggunakan gambar Anda sendiri, asalkan resolusinya sama dengan milik saya.

Selanjutnya, kembali ke folder "Is-node-js-pasta/views".

Di sini, buatlah file kosong bernama:

- functions.ejs
- 404.ejs
- home.ejs
- manage.ejs
- paste.ejs
- trends.ejs

Selanjutnya buka folder "Is-node-js-pasta" dan buat file kosong bernama "index.js".

Jika Anda masih tidak yakin apakah penempatannya benar, silakan dicek di source code buku ini.

Menginstall Package NPM di Direktori Project

Sekarang kita sudah menempatkan package non NPM di folder yang benar.

Saatnya menginstall package NPM-nya.

Masih di folder "ls-node-js-pasta", jalankan perintah ini:

```
npm init
```

Selanjutnya, isi saja sesuai yang ditawarkan hingga jawaban terakhir (yes/no).

Kemudian, ketikkan perintah ini:

```
npm install body-parser --save
npm install express --save
npm install ejs --save
npm install randomstring --save
npm install sqlite3 --save
```

Dari langkah tersebut, akan dibuat file baru secara otomatis bernama package.json.

Pastikan isi filenya sama dengan ini:

```
{
  "name": "ls-node-js-pasta",
  "version": "1.0.0",
  "description": "Node.js Pasta adalah Node.js Pastebin Clone",
  "main": "index.js",
  "scripts": {
    "start": "node index.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Anonim",
  "license": "UNLICENSED",
  "dependencies": {
    "body-parser": "^1.17.2",
    "ejs": "^2.5.7",
    "express": "^4.15.4",
    "randomstring": "^1.1.5",
    "sqlite3": "^3.1.9"
  }
}
```

Perhatikan pada bagian yang digarisbawahi.

Pastikan ada bagian tersebut.

Nama author bisa beda, tergantung apa yang kita isi saat melakukan npm init. Begitu pula license.

Mengimpor Package NPM

Setelah package NPM ter-install, kita bisa mengimpornya di dalam script.

Sekarang buka file "index.js" pada folder "ls-node-js-pasta".

Saat ini file tersebut masih kosong.

Isi file tersebut dengan kode ini:

```
var express = require('express');
var app = express();
var bodyParser = require('body-parser');
var randomstring = require("randomstring");
var sqlite3 = require('sqlite3').verbose();
var fs = require('fs');
```

Di sinilah kita mengimpor package-package NPM agar nanti bisa digunakan.

Perhatikan pada bagian:

```
var app = express();
```

Mengapa di bagian tersebut ada fungsi express()?

Padahal yang lain tidak ada hal semacam itu.

Hal itu disebabkan pada saat mengimport express di bagian:

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

Yang diimport merupakan fungsi, karena itu harus dipanggil dengan express().

Mempersiapkan Framework Expressjs

Setelah mengimpor package-package NPM ke script "index.js", kita akan mempersiapkan framework Expressjs.

Disini kita akan membuat kerangka routes dan mengonfigurasi Expressjs agar bisa berjalan semestinya.

Buka file "index.js" kemudian di bawah kode impor NPM ini:

```
var express = require('express');
var app = express();
var bodyParser = require('body-parser');
var randomstring = require("randomstring");
var sqlite3 = require('sqlite3').verbose();
var fs = require('fs');
```

, tambahkan kode ini:

```
app.set('view engine', 'ejs'); //YANG INI - 1
app.set('view options', { //YANG INI - 2
  rmWhitespace: true
});

app.use('/views', express.static(__dirname + '/views/')); //YANG INI - 3
app.use(bodyParser.urlencoded({ extended: true })); //YANG INI - 4

app.get('/', function (req, res) { //YANG INI - 5

});

app.get('/trends', function (req, res) {

});

app.post('/new-paste', function(req, res) {

});

app.post('/manage/save-paste/:manageID', function(req, res) { //YANG INI - 6

});
```



```
app.post('/manage/delete-paste/:manageID', function(req, res) {  
  });  
  
app.get('/manage/:manageID', function (req, res) {  
  })  
  
app.get('/paste/:pasteID', function (req, res) {  
  })  
  
app.get('*', function(req, res){ //YANG INI - 7  
  });  
  
app.set('port', (process.env.PORT || 5000)); //YANG INI - 8  
  
app.listen(app.get('port'), function () { //YANG INI - 9  
  console.log('Node app is running on port', app.get('port'));  
  });
```

Perhatikan kode bagian "//YANG INI - 1".

Di sana, kita menge-set EJS sebagai view engine.

Artinya setelah kita mengimpor EJS, kita harus memberitahu Express.js bahwa kita menggunakan template engine tersebut.

Pada kode bagian "//YANG INI - 2", kita mengonfigurasi EJS agar menghapus whitespace.

Bisa dilihat di dokumentasinya pada (<https://github.com/mde/ejs#options>)

Jika hal itu tidak dilakukan, saat kita membuat paste akan ada tambahan whitespace yang tidak diinginkan.

Pada kode bagian "//YANG INI - 3", kita memberi akses pada Express.js agar dapat merespon file statis.

Yang dimaksud file statis di sini adalah file CSS, HTML, Gambar, Javascript, dan file

apapun yang terletak pada direktori yang didaftarkan pada fungsi:

```
app.use('/views', express.static(__dirname + '/views/'));
```

Jadi, apabila kita tidak melakukan ini, request semacam ini:

```
http://nama_host/views/js/jquery.min.js
```

Tidak sah dan file "jquery.min.js" tersebut tidak bisa diakses.

Agar bisa diakses, kita harus melakukan "//YANG INI - 3".

Pada kode bagian "//YANG INI - 4":

```
app.use(bodyParser.urlencoded({ extended: true }));
```

Kita memberitahu Expressjs untuk menggunakan module Body Parser untuk mem-parse application/x-www-form-urlencoded.

Jadi apabila kita memiliki form pada sebuah file HTML, dan pada form tersebut ada input dengan atribut name="inputpertama", seperti ini:

```
<input type="text" name="inputpertama">
```

Maka di script servernya nilai input tadi bisa diakses dengan:

```
app.post('/getinput', function(req, res) {  
  console.log(req.body.inputpertama );//bisa begini...  
});
```

Tidak hanya untuk merequest "/getinput" saja, bisa juga "/" atau "/apapun" tergantung apa yang ditulis pada argument app.post.

Anda bisa lihat pada source code buku ini bahwa terdapat beberapa penggunaan Body Parser semacam ini.

Sekarang perhatikan pada bagian "//YANG INI - 5":

```
app.get('/', function (req, res) {  
});
```

Dengan kode ini, kita akan merespon request "/" atau yang biasa disebut Home.

Request semacam ini bisa juga disertai dengan variabel seperti pada bagian "//YANG INI - 6".

```
app.post('/manage/save-paste/:manageID', function(req, res) {  
});
```

Perhatikan bahwa variabel diletakkan setelah "/manage/save-paste", yakni ":manageID".

Tanda titik dua pada routing di Express.js menandakan bahwa itu adalah variabel. Dengan cara tersebut, kita bisa mengakses:

```
http://nama_host/manage/save-paste/123  
http://nama_host/manage/save-paste/345  
http://nama_host/manage/save-paste/atauapapunjuga
```

Sekarang perhatikan pada kode bagian "//YANG INI - 7":

```
app.get('*', function(req, res){  
});
```

Di situ ada karakter asterisk (*).

Itu menandakan bahwa request apapun yang tidak terdaftar akan diarahkan ke situ.

Routes semacam ini berguna untuk mengarahkan halaman yang tidak ditemukan (404) ke halaman 404 yang telah disiapkan.

Pada kode bagian "//YANG INI - 8":

```
app.set('port', (process.env.PORT || 5000));
```

Kita memberitahu Express.js bahwa server akan berjalan pada port 5000 atau pada port yang didefinisikan oleh environment variable (ditandai dengan process.env.PORT).

Hal ini berguna saat kita men-deploy aplikasi ini ke Heroku karena Heroku sendiri

yang menentukan port berapa untuk aplikasi kita walaupun setelah di-deploy semua aplikasi Heroku berjalan pada port 443 secara default.

Hal tersebut disebabkan (sepertinya karena) Heroku menggunakan semacam proxy untuk mengarahkan subdomain kita di port 443 ke port yang asli di network internal mereka.

URL dengan port yang asli tadi tidak bisa diakses secara langsung dari internet.

Dan akhirnya, pada kode bagian "//YANG INI - 9":

```
app.listen(app.get('port'), function () {  
    console.log('Node app is running on port', app.get('port'));  
});
```

Kita memerintahkan Express.js untuk menjalankan server pada port tadi.

Membuat Database dengan SQLite3

Karena aplikasi ini bisa menyimpan paste, maka diperlukan sebuah DBMS.

Mengingat akun Heroku yang free tidak menyediakan fitur MySQL, maka saya memilih SQLite3 sebagai DBMS.

SQLite3 dapat digunakan di Heroku karena hanya berupa file dan tidak butuh server yang terpisah.

Memang ada kekurangannya, tapi untuk skala kecil SQLite3 masih dapat digunakan.

Sekarang buka kembali file "index.js".

Di bawah script untuk import module:

```
var express = require('express');  
var app = express();  
var bodyParser = require('body-parser');  
var randomstring = require("randomstring");  
var sqlite3 = require('sqlite3').verbose();  
var fs = require('fs');
```

Tulis kode ini:

```

var db;

db = new sqlite3.Database('./db_pasta.db');//YANG INI - 1
db.run();//YANG INI - 2
`CREATE TABLE IF NOT EXISTS pastes (
    id INTEGER PRIMARY KEY,
    newpaste TEXT,
    syntaxhi TEXT,
    pasteex TEXT,
    pasteexpos TEXT,
    title TEXT,
    pasteperm TEXT,
    pastemanagerperm TEXT,
    hits INTEGER
);`, function(){
    db.each("SELECT * from pastes", function(err, row) { //YANG INI - 3
        console.log("scheduling the timeout for: " + row.title);
        setTimeout(function(){ //YANG INI - 4
            console.log("deleting the row automatically...");
            db.run("DELETE FROM pastes WHERE id = '" + row.id + "'");
            console.log("deleted the row automatically...");
        }, textToMS(row.pasteex));
    });
});

```

Bila Anda tidak yakin dengan penempatannya, silakan cek file "index.js" pada source code buku ini.

Bila kita perhatikan pada script di atas, database aplikasi ini sangat sederhana.

Hanya memiliki satu tabel.

Memang demikian. Karena saya berusaha menyederhanakan clone Pastebin agar mudah dipelajari.

Jika Anda memiliki versi sendiri yang lebih kompleks tidak masalah. Anda tinggal kembangkan sendiri aplikasi ini.

Sekarang perhatikan pada kode bagian "//YANG INI - 1".

Di sana kita membuat database baru atau meng-overwrite database lama dengan nama " db_pastebin_clone.db".

File database itu berada di folder "ls-node-js-pasta".

Selanjutnya, pada kode bagian "//YANG INI - 2", kita menjalankan query:

```
CREATE TABLE IF NOT EXISTS pastes (  
  id INTEGER PRIMARY KEY,  
  newpaste TEXT,  
  syntaxhi TEXT,  
  pasteex TEXT,  
  pasteexpos TEXT,  
  title TEXT,  
  pasteperm TEXT,  
  pastemanagerperm TEXT,  
  hits INTEGER  
);
```

Kolom "id" adalah id dari row. Di sana tertulis "INTEGER PRIMARY KEY".

Hal itu dilakukan agar "id" bersifat auto increment.

Kolom "newpaste" menyimpan teks yang kita paste pada aplikasi ini.

Kolom "syntaxhi" menyimpan setting syntax highlighting apa yang kita gunakan untuk paste tersebut.

Kolom "pasteex" menyimpan setting mengenai berapa lama paste disimpan dalam database hingga expired dan dihapus secara otomatis.

Kolom "pasteexpos" menyimpan setting mengenai apakah paste ini bersifat public atau unlisted.

Kolom "title" adalah judul dari paste.

Kolom "pasteperm" adalah permalink dari paste yang bisa di-share dan dilihat publik.

Kolom "pastemanagerperm" adalah permalink dari paste manager yang hanya ditujukan untuk pembuat paste. Ini sebaiknya tidak dibagikan pada orang lain. Selain itu, tidak ada cara untuk menemukan permalink ini jika pembuat paste lupa.

Kolom "hits" mencatat total kunjungan pada paste. Bukan paste manager.

Nilai "hits" akan bertambah seiring bertambahnya kunjungan pada paste.

Nilai itu juga menjadi faktor penentu urutan listing dari paste-paste pada halaman "Trends".

Pada fungsi db.run tadi, kita juga memberikan callback:

```
db.each("SELECT * from pastes", function(err, row) { //YANG INI - 3
    console.log("scheduling the timeout for: " + row.title);
    setTimeout(function(){ //YANG INI - 4
        console.log("deleting the row automatically...");
        db.run("DELETE FROM pastes WHERE id = '" + row.id + "'");
        console.log("deleted the row automatically...");
    }, textToMS(row.pasteex));
});
```

Pada kode bagian "//YANG INI - 3", kita menjelajahi tiap row dari tabel paste dan setiap satu row ditemukan, maka callback ini dijalankan:

```
console.log("scheduling the timeout for: " + row.title);
```

```
setTimeout(function(){ //YANG INI - 4
    console.log("deleting the row automatically...");
    db.run("DELETE FROM pastes WHERE id = '" + row.id + "'");
    console.log("deleted the row automatically...");
}, textToMS(row.pasteex));
```

Di sini kita akan menjadwalkan penghapusan setiap row dari tabel paste selama:

```
textToMS(row.pasteex)
```

Jadi setTimeout tadi akan selama itu.

Kita menggunakan textToMS karena kita akan mengonversi 1 Day, 12 Hours ... 2 Minutes menjadi jumlah milisecond dalam integer.

Kita melakukan ini pada saat aplikasi ini baru dijalankan karena boleh jadi ketika aplikasi ini berhenti entah karena crash atau faktor lain, masih ada paste yang belum dihapus karena belum memenuhi durasi lifetime-nya.

Sebenarnya cara ini kurang akurat, tetapi setidaknya memberi ide mengenai bagaimana cara membuat expirable content pada sebuah aplikasi web.

Membuat Fungsi-Fungsi Utility

Setelah kita membuat database untuk Node.js Pasta, kita akan membuat fungsi-fungsi utility-nya.

Fungsi-fungsi yang akan kita buat ini berguna untuk menyederhanakan penulisan kode sehingga tidak banyak copy paste dari satu baris ke baris yang lain.

Adapun fungsi-fungsi utility yang akan kita buat adalah:

- textToMS
- getBaseURL
- getTheExcerpt

Fungsi textToMS dan getBaseURL ditulis di file "index.js" sedangkan getTheExcerpt akan ditulis pada template "functions.ejs".

Fungsi getTheExcerpt ditulis di file template karena ini merupakan utility untuk keperluan template rendering.

Fungsi textToMS

Pertama-tama, buka file "index.js", kemudian scroll hingga bagian terbawah.

Selanjutnya tulis kode ini:

```
function textToMS(textTime){
  if(textTime == "1 Day"){
    return 24 * 60 * 60 * 1000;
  } else if(textTime == "12 Hours") {
    return 12 * 60 * 60 * 1000;
  } else if(textTime == "3 Hours") {
    return 3 * 60 * 60 * 1000;
  } else if(textTime == "30 Minutes") {
    return 30 * 60 * 1000;
  } else if(textTime == "2 Minutes") {
    return 2 * 60 * 1000;
  }
}
```

Fungsi di atas digunakan untuk mengonversi teks "1 Day, 12 Hours, 3 Hours, 30 Minutes, dan 2 Minutes" menjadi integer milisecond.

Hal ini dilakukan karena pada HTML, nilai ini disajikan dalam sebuah dropdown yang value-nya berupa string.

Mengenai rumusnya cukup jelas. 1 Day misalnya, adalah 24 jam x 60 menit x 60 detik x 1000 milidetik.

Sisanya tidak jauh beda. Silakan tafsirkan sendiri.

Fungsi getBaseURL

Sekarang, di bawah fungsi textToMS pada file "index.js", tulis kode ini:

```
function getBaseURL(req){
  var port = app.get('port');
  var trailing = port == 80 || port == 443 ? "" : (":" + port);
  return (req.secure?'https://':'http://') + req.hostname + trailing;

  //Jika Anda men-deploy aplikasi ini di heroku, replace kode
  //di atas dengan ini:
  //return "https://namasubdomainanda.herokuapp.com";
  //misalnya (yang punya saya):
  //return "https://ls-node-js-pasta.herokuapp.com";
}
```

Fungsi ini digunakan untuk mendapatkan base URL dari aplikasi ini.

Fungsi ini membutuhkan argument "req" yang didapatkan pada saat routing, misalnya:

```
app.get('/manage/:manageID', function (req, res) {
  console.log(getBaseURL(req));
})
```

Tetapi itu tidak mutlak diperlukan jika Anda men-deploy-nya ke Heroku seperti yang dijelaskan pada komentar di fungsi tersebut:

```
//Jika Anda men-deploy aplikasi ini di heroku, replace kode
//di atas dengan ini:
//return "https://namasubdomainanda.herokuapp.com";
//misalnya (yang punya saya):
//return "https://ls-node-js-pasta.herokuapp.com";
```

Jika Anda menjalankan aplikasi ini di localhost, Anda tidak perlu mengikuti komentar itu.

Saya memilih cara manual untuk mendapatkan base URL di Heroku, karena cara ini adalah yang paling mudah mengingat Heroku sepertinya menggunakan proxy untuk mengakses server yang sebenarnya.

Fungsi getBaseURL ini akan diperlukan ketika kita membuka halaman "Paste Manager" di mana di sana aplikasi ini akan meng-generate URL untuk mengakses halaman "Paste Manager" dan halaman "Paste" yang telah dibahas pada subbab "Rancangan User Interface".

The screenshot displays the Node.js Pasta web application. At the top, there's a header with 'Node.js Pasta' and a 'New Paste' button. The main area is titled 'This Paste' and contains a code editor with the following JavaScript code:

```
/*
trik-10.js
Membuat CSV Report tentang Ranking Website di Google
*/
var Nightmare = require('nightmare');
var nightmare = Nightmare({
  show: true
});
var fs = require('fs');
var csv = require('fast-csv');
```

Below the code editor is a 'Settings' panel with the following options:

- Syntax Highlighting: text
- Paste Expiration: 1 Day
- Paste Exposure: Public
- Title: scraper-1.js

At the bottom of the settings panel are 'Save Paste' and 'Delete Paste' buttons. To the right of the settings is a 'Links for This Paste' panel, which is highlighted with a red box. It contains two fields:

- Manager URL: <http://localhost:5000/manage/utudybd> (with a 'Copy!' button)
- Paste URL: <http://localhost:5000/paste/tcynioc> (with a 'Copy!' button)

A red arrow points from the 'Syntax Highlighting' dropdown in the settings panel to the 'Manager URL' field in the links panel. The footer of the application reads 'Copyright © Luslikar Sheba 2017'.

URL yang Digenerate

Fungsi getTheExcerpt

Fungsi ini agak spesial.

Fungsi ini ditulis di file template (yang berekstensi .ejs).

Fungsi ini ditulis di file tersebut karena ini merupakan fungsi utility untuk template rendering.

Sekarang, buka file "functions.ejs", lalu tulis kode ini:

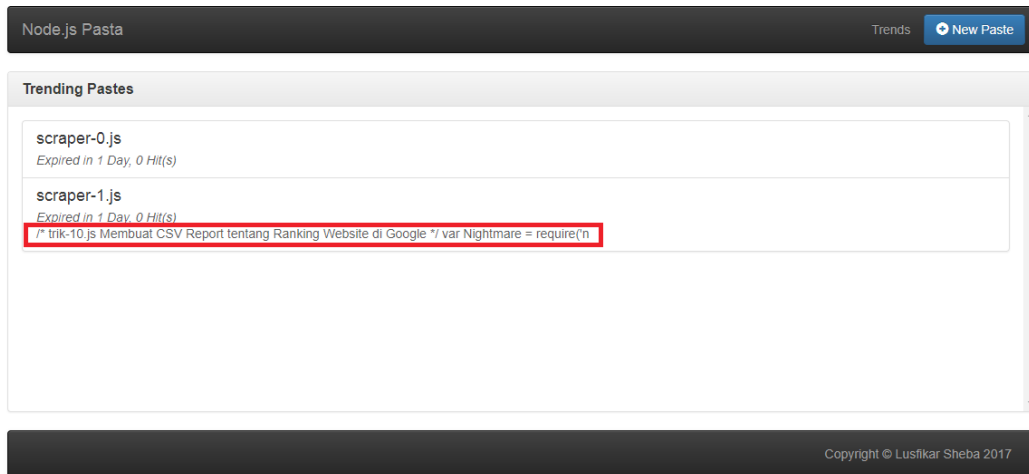
```
<%  
getTheExcerpt = function(wholeText){  
    var strLen = wholeText.length;  
    if(strLen < 100){  
        return wholeText.substring(0, strLen);  
    }  
    return wholeText.substring(0, 100);  
}  
%>
```

Yang dilakukan oleh fungsi ini adalah mengambil potongan teks dari sebuah teks penuh.

Tadi kita lihat pada bagian database bahwa kolom "newpaste" dari tabel "pastes" menyimpan teks penuh dari paste yang kita buat.

Padahal pada halaman trends kita hanya memerlukan potongan kecil dari teks tersebut.

Oleh karena itulah kita menggunakan fungsi ini, sehingga yang tampil pada row di halaman "Trends" adalah hanya potongannya seperti ini:



Excerpt di Halaman "Trends"

Membuat Halaman 404 Not Found

Sekarang kita telah menyelesaikan kira-kira 50 persen dari seluruh script dalam project ini.

Mungkin Anda masih ingat routing di file "index.js" yang masih belum diisi dengan response.

Mulai sekarang kita akan mengisinya sambil membuat user interfacenya dalam HTML, CSS, dan Javascript.

Perlu saya ingatkan bahwa mulai saat ini, sering-seringlah mengecek kesesuaian kode di buku ini dengan yang ada dalam source code buku ini.

Pertama-tama, buka file "404.ejs" dalam folder "ls-node-js-pasta/views".

File ini masih kosong, tetapi pada folder yang sama di dalam **SOURCE CODE BUKU INI** sudah ada isinya.

Langsung copy-paste saja dari source code tadi ke file "404.ejs" yang masih kosong ini.

Isinya kira-kira begini:

```
<!DOCTYPE html>
<html lang="en">
```

```

<head>
<meta charset="utf-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1">

<title>404: ERROR NOT FOUND - Node.js Pasta</title>

<meta name="description" content="Node.js Pasta - Content Not Found">
<meta name="author" content="Anonim">

<!-- YANG INI - 1 -->
<link rel="shortcut icon" type="image/png" href="/views/img/favicon.png"/>
<link href="/views/css/bootstrap.min.css" rel="stylesheet">
<link href="/views/css/bootstrap-theme.min.css" rel="stylesheet">
<script src="/views/js/jquery-3.2.1.min.js"></script>
<script src="/views/js/bootstrap.min.js"></script>
</head>
<body>

<div class="container">
<nav class="navbar navbar-default navbar-inverse">
<div class="container-fluid">
<div class="navbar-header">
<button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-
target="#navbar" aria-expanded="false" aria-controls="navbar">
<span class="sr-only">Toggle navigation</span>
<span class="icon-bar"></span>
<span class="icon-bar"></span>
<span class="icon-bar"></span>
</button>
<a class="navbar-brand" href="/">Node.js Pasta</a>
</div>
<div id="navbar" class="navbar-collapse collapse">
<ul class="nav navbar-nav navbar-right">
<li><a href="/trends">Trends</a></li>
<li><div class="btn-nav"><a class="btn btn-primary btn-small navbar-btn"
href="/"><span class="glyphicon glyphicon-plus-sign"></span> New Paste</a></li>
</ul>
</div>
</div>
</nav>

<div class="row">

```

```

<div class= "col-lg-12 col-md-12 col-sm-12 col-xs-12">
<div class="jumbotron">
<h1>404: ERROR NOT FOUND</h1>
<p>not found not found not found not found not found not found not found not found not found
not found .</p>
<p>not found not found not found not found not found not found not found not found not found
not found .</p>
<p>not found not found not found not found not found not found not found not found not found
not found .</p>
</div>
</div>
</div>
<nav class="navbar navbar-default navbar-inverse">
<div class="container-fluid">
<p class="navbar-text pull-right">Copyright &#169; Anonim 2017</p>
</div>
</nav>
</body>
</html>

```

Perhatikan pada kode bagian "//YANG INI - 1".

Di situ kita mengimpor package-package non NPM, yaitu:

- Bootstrap
- JQuery

Di situ kita juga mendaftarkan file gambar "favicon.png" pada folder "ls-node-js-pasta/views/img/" agar muncul di tab di browser.

Halaman ini akan muncul apabila user mengakses URL yang salah.

Agar bisa seperti itu, buka file "index.js" versi Anda (bukan yang dari source code buku ini), kemudian fokuskan perhatian Anda pada route ini:

```

app.get('*', function(req, res){
});

```

Kemudian tambahkan kode-kode ini di dalam route tersebut:

```

res.status(404);
res.render('404');

```

Sehingga menjadi:

```
app.get('*', function(req, res){  
  res.status(404);  
  res.render('404');  
});
```

Dengan demikian, URL apapun yang tidak terdaftar di route akan diarahkan ke halaman "404".

Fungsi `res.status` adalah HTTP Status Code yang diberikan pada route tersebut.

Daftar lengkapnya ada di:

<https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html>

Sedangkan fungsi `res.render` digunakan untuk menampilkan template "404.ejs" sebagai HTML di sisi browser pengguna.

Jadi di sini tidak perlu ditulis lengkap dengan ekstensinya (".ejs").

Sekarang Anda bisa coba jalankan aplikasi ini.

Jalankan perintah ini di command line (dengan titik di sebelah node):

```
node .
```

Kemudian buka browser anda ke:

<http://localhost:5000>

Maka nanti halaman 404 akan ditampilkan.

Membuat Halaman Home (New Paste)

Setelah membuat halaman 404, sekarang kita membuat halaman "Home".

Pada halaman ini, kita akan menampilkan form untuk membuat halaman pembuatan paste.

Buka file kosong "home.ejs" yang telah kita buat sebelumnya.

Halaman ini masih kosong, maka copy dari source code buku ini dari file yang sama (ada di folder views).

Maka isinya akan seperti ini:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1">

<title>Node.js Pasta</title>

<meta name="description" content="Node.js Pasta. Share your pastes here!">
<meta name="author" content="Anonim">

<!-- [ Social Media meta tag ] -->
<meta content='summary' name='twitter:card' />
<meta content='@ciyy_space' name='twitter:site' />
<meta content='@ciyy_space' name='twitter:creator' />

<meta content='<%=hostname%>' property='og:url' />
<meta content='Node.js Pasta' property='og:site_name' />
<meta content='Node.js Pasta' property='og:title' />
<meta content='Node.js Pasta' name='twitter:title' />
<meta content='website' property='og:type' />
<meta content='<%=hostname%>/views/img/ciyy-200x200.png' property='og:image' />
<meta content='<%=hostname%>/views/img/ciyy-200x200.png' name='twitter:image' />
<meta content='Node.js Pasta. Share your pastes here!' property='og:description' />
<meta content='Node.js Pasta. Share your pastes here!' name='twitter:description' />
<meta content='en_US' property='og:locale' />
<!-- [/ Social Media meta tag ] -->

<link rel="shortcut icon" type="image/png" href="/views/img/favicon.png"/>
<link href="/views/css/bootstrap.min.css" rel="stylesheet">
<link href="/views/css/bootstrap-theme.min.css" rel="stylesheet">
<script src="/views/js/jquery-3.2.1.min.js"></script>
<script src="/views/js/bootstrap.min.js"></script>
</head>
<body>
<% include ./functions %>
<div class="container">
<nav class="navbar navbar-default navbar-inverse">
<div class="container-fluid">
```



```

<div class="navbar-header">
<button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-
target="#navbar" aria-expanded="false" aria-controls="navbar">
<span class="sr-only">Toggle navigation</span>
<span class="icon-bar"></span>
<span class="icon-bar"></span>
<span class="icon-bar"></span>
</button>
<a class="navbar-brand" href="/">Node.js Pasta</a>
</div>
<div id="navbar" class="navbar-collapse collapse">
<ul class="nav navbar-nav navbar-right">
<li><a href="/trends">Trends</a></li>
<li><div class="btn-nav"><a class="btn btn-primary btn-small navbar-btn"
href="/"><span class="glyphicon glyphicon-plus-sign"></span> New Paste</a></li>
</ul>
</div>
</div>
</nav>

<div class="row">
<div class="col-lg-12 col-md-12 col-sm-12 col-xs-12">
<div class="jumbotron">
<h1>Apa itu Node.js Pasta?</h1>
<p>Node.js Pasta adalah alat untuk berbagi teks yang di-paste-kan ke website ini.</p>
<p>Anda bisa membuat, mengubah, dan menghapus paste Anda TANPA LOGIN di sini!</p>
<p>Jika berminat dengan website semacam ini, <a
href="https://projects.co.id/public/browse_users/view/99bc11/ciyyospace"
target="_blank" rel="nofollow">pekerjaan saya di sini</a> atau <a
href="javascript:void(0)" target="_blank" rel="nofollow">beli bukunya di
sini</a>.</p>
</div>
</div>
</div>

<form action="/new-paste" method="post">
<div class="row">
<div class="col-lg-12 col-md-12 col-sm-12 col-xs-12">
<div class="panel panel-default">
<div class="panel-heading">
<h3 class="panel-title"><b>New Paste</b></h3>
</div>
<div class="panel-body">

```

```

<textarea id="pc-ta-newpaste" name="pcTaNewpaste" class="form-control"
rows="10"></textarea>
</div>
</div>
</div>
</div>

<div class="row">
<div class="col-lg-6 col-md-6 col-sm-6 col-xs-6">
<div class="panel panel-default">
<div class="panel-heading">
<h3 class="panel-title"><b>Settings</b></h3>
</div>
<div class="panel-body">
<div class="form-group">
<label for="pc-sl-syntaxhi">Syntax Highlighting:</label>
<select id="pc-sl-syntaxhi" name="pcSlSyntaxhi" class="form-control">
<%
var options = [ "text", "html", "css", "javascript" ];
for ( var i = 0; i < options.length; i++ )
{
%><option value="<%=options[i] %>"><%=options[i] %></option><%
}
%>
</select>
<label for="pc-sl-pasteex">Paste Expiration:</label>
<select id="pc-sl-pasteex" name="pcSlPasteex" class="form-control">
<%
var options = [ "1 Day", "12 Hours", "3 Hours", "30 Minutes", "2 Minutes" ];
for ( var i = 0; i < options.length; i++ )
{
%><option value="<%=options[i] %>"><%=options[i] %></option><%
}
%>
</select>

<label for="pc-sl-pasteexpos">Paste Exposure:</label>
<select id="pc-sl-pasteexpos" name="pcSlPasteexpos" class="form-control">
<%
var options = [ "Public", "Unlisted" ];
for ( var i = 0; i < options.length; i++ )
{
%><option value="<%=options[i] %>"><%=options[i] %></option><%

```

```

}
%>
</select>

<label for="pc-tx-title">Title:</label>
<input id="pc-tx-title" type="text" name="pcTxTitle" class="form-control">
<br/>
<input id="pc-sb-newpaste" name="pcSbNewpaste" class="btn btn-danger btn-block"
type="submit" value="Create New Paste" />
</div>
</div>
</div>
</div>

<div class= "col-lg-6 col-md-6 col-sm-6 col-xs-6">
<div class="panel panel-default">
<div class="panel-heading">
<h3 class="panel-title"><b>Recent Pastes</b></h3>
</div>
<div class="panel-body" style="min-height: 336px; max-height: 336px; overflow-y:
scroll;">
<div class="list-group">
<% if (locals.therows) { %>
<% therows.forEach(function(singleRow) { %>
<a href="<%= hostname %>/paste/<%= singleRow.pasteperm %>" target="_blank"
class="list-group-item">
<h4 class="list-group-item-heading"><%= singleRow.title %></h4>
<p class="list-group-item-text"><i>Expired in <%= singleRow.pasteex %>, <%=
singleRow.hits %> Hit(s)</i></p>
<p class="list-group-item-text"><%= getTheExcerpt(singleRow.newpaste) %></p>
</a>
<% }); %>
<% } %>
</div>
</div>
</div>
</div>
</div>
</div>
</form>
<nav class="navbar navbar-default navbar-inverse">
<div class="container-fluid">
<p class="navbar-text pull-right">Copyright &#169; Anonim 2017</p>
</div>

```

```
</nav>
</body>
</html>
```

Di sana ada beberapa baris kode yang bentuknya seperti ini:

```
<select id="pc-sl-syntaxhi" name="pcSlSyntaxhi" class="form-control">
<%
var options = [ "text", "html", "css", "javascript" ];
for ( var i = 0; i < options.length; i++ )
{
%><option value="<%=options[i] %>"><%=options[i] %></option><%
}
%>
</select>
```

Kode tersebut tugasnya adalah menampilkan item dari sebuah dropdown.

Yang barusan itu untuk dropdown jenis syntax highlighting yang pilihannya "text", "html", "css", dan "javascript".

Nanti string-string tersebut akan diinjeksikan ke variabel javascript pada halaman "Paste" setelah disimpan dalam database.

Sekarang di sisi server, ingat kembali tentang route ini:

```
app.get('/', function (req, res) { //YANG INI - 5
});
```

Ubah jadi:

```
app.get('/', function (req, res) {
    db.all("SELECT * FROM pastes WHERE pasteexpos='Public' LIMIT 20", function(err,
rows) {
        res.render('home', {
            therows : rows, //YANG INI - 1
            hostname: getBaseUrl(req) //YANG INI - 2
        });
    });
});
```

Di sini, ketika user mengakses halaman "Home", maka server akan menarik data dari database.

Data yang diambil adalah:

```
SELECT * FROM pastes WHERE pasteexpos='Public' LIMIT 20
```

Yang artinya hanya semua kolom dari tabel pastes yang paste exposurenya "Public" dan batasi hanya 20 row.

Di sini kita menggunakan db.all jadi yang diterima pada rows di argument callbacknya adalah semua hasil dari query di atas. Bukan satu per satu seperti db.each.

Selanjutnya saat query tersebut selesai dieksekusi, callback dipanggil dan pada akhirnya server merender halaman "Home" dengan menginjeksikan rows sebagai therows (//YANG INI - 1) dan base URL sebagai hostname (//YANG INI - 2).

Maka nilai ini pada file "home.ejs":

```
<meta content='<%=hostname%>' property='og:url' />
```

Akan di-replace dengan hostname dari injeksi di server tadi.

Sekarang anda bisa coba jalankan aplikasi ini:

```
node .
```

Dan buka browser ke:

```
http://localhost:5000
```

Nanti halaman 404 akan digantikan halaman "Home" lengkap dengan form-nya.

Membuat Halaman Manage

Setelah selesai membuat halaman "Home" dan route-nya, kita akan membuat halaman "Manage".

Di halaman ini, user akan bisa menampilkan paste yang telah dibuat di halaman "Home" dan mengedit atau menghapusnya.

Di sini juga akan di-generate URL manager dan URL paste

Pertama-tama bukanlah file "manage.ejs" dari folder "ls-node-js-pasta/views".

Kira-kira isinya seperti ini:

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="utf-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1">

  <title>Manage <%= therows[0].title %> - Node.js Pasta</title>

  <meta name="description" content="Manage <%= therows[0].title %> with Node.js
Pasta">
  <meta name="author" content="Anonim">

  <!-- [ Social Media meta tag ] -->
  <meta content='summary' name='twitter:card' />
  <meta content='@ciyy_space' name='twitter:site' />
  <meta content='@ciyy_space' name='twitter:creator' />

  <meta content='<%=hostname%>/manage/<%=therows[0].pastemanagerperm %>'
property='og:url' />
  <meta content='Manage <%= therows[0].title %> - Node.js Pasta'
property='og:site_name' />
  <meta content='Manage <%= therows[0].title %> - Node.js Pasta'
property='og:title' />
  <meta content='Manage <%= therows[0].title %> - Node.js Pasta'
name='twitter:title' />
  <meta content='article' property='og:type' />
  <meta content='<%=hostname%>/views/img/ciyy-200x200.png' property='og:image' />
  <meta content='<%=hostname%>/views/img/ciyy-200x200.png' name='twitter:image' />
  <meta content='Manage <%= therows[0].title %> with Node.js Pasta'
property='og:description' />
  <meta content='Manage <%= therows[0].title %> with Node.js Pasta'
name='twitter:description' />
  <meta content='en_US' property='og:locale' />
  <!-- [ / Social Media meta tag ] -->

  <link rel="shortcut icon" type="image/png" href="/views/img/favicon.png" />
```

```

    <!-- Latest compiled and minified CSS -->
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/
bootstrap.min.css"
        integrity="sha384-BVYiiSIFeK1dGmJRAkycuHAHRg320mUcww7on3RYdg4Va+PmSTsz/
K68vbdEjh4u" crossorigin="anonymous">

    <!-- Optional theme -->
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/
bootstrap-theme.min.css"
        integrity="sha384-
rHyoN1iRsVXV4nD0JutlnGaslCJuC7uwjduW9SVrLvRYooPp2bWYgmgJQIXw1/Sp"
crossorigin="anonymous">

    <script src="https://code.jquery.com/jquery-3.3.1.min.js"
        integrity="sha256-FgpCb/KJQlLNfOu91ta32o/NMZxltwRo8QtmkMRdAu8="
crossorigin="anonymous"></script>

    <!-- Latest compiled and minified JavaScript -->
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"
        integrity="sha384-
Tc5Iqib027qvyjSMfHj0MaLkfuWVxZxUPnCJA7l2mCWNIpG9mGCD8wGNICPD7Txa"
crossorigin="anonymous">
    </script>

    <script src="https://cdn.jsdelivr.net/npm/clipboard@2/dist/clipboard.min.js"></
script>
    <!--
    <link href="/views/css/bootstrap.min.css" rel="stylesheet">
    <link href="/views/css/bootstrap-theme.min.css" rel="stylesheet">
    <script src="/views/js/jquery-3.2.1.min.js"></script>
    <script src="/views/js/bootstrap.min.js"></script>
    -->
</head>

<body>

    <div class="container">
        <nav class="navbar navbar-default navbar-inverse">
            <div class="container-fluid">
                <div class="navbar-header">
                    <button type="button" class="navbar-toggle collapsed" data-
toggle="collapse" data-target="#navbar">

```

```

        aria-expanded="false" aria-controls="navbar">
        <span class="sr-only">Toggle navigation</span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
    </button>
    <a class="navbar-brand" href="/">Node.js Pasta</a>
</div>
<div id="navbar" class="navbar-collapse collapse">
    <ul class="nav navbar-nav navbar-right">
        <li><a href="/trends">Trends</a></li>
        <li>
            <div class="btn-nav"><a class="btn btn-primary btn-small
navbar-btn" href="/"><span
                                class="glyphicon glyphicon-plus-sign"></span>
New Paste</a>
            </li>
        </ul>
    </div>
</div>
</nav>

<form method="post">
    <div class="row">
        <div class="col-lg-12 col-md-12 col-sm-12 col-xs-12">
            <div class="panel panel-default">
                <div class="panel-heading">
                    <h3 class="panel-title"><b>This Paste</b></h3>
                </div>
                <div class="panel-body">
                    <textarea id="pc-ta-newpaste" name="pcTaNewpaste"
class="form-control" rows="10">
                        <%= therows[0].newpaste %>
                    </textarea>
                </div>
            </div>
        </div>
    </div>

    <div class="row">
        <div class="col-lg-6 col-md-6 col-sm-6 col-xs-6">
            <div class="panel panel-default">
                <div class="panel-heading">

```



```

        <h3 class="panel-title"><b>Settings</b></h3>
    </div>
    <div class="panel-body">
        <div class="form-group">
            <label for="pc-sl-syntaxhi">Syntax
Highlighting:</label>
            <select id="pc-sl-syntaxhi" name="pcSlSyntaxhi"
class="form-control">
                <%
                var options = [ "text", "html", "css",
"javascript" ];
                for ( var i = 0; i < options.length; i++ )
                {
                    var selected = ( therows[0].syntaxhi ==
options[i] ) ? "selected" : "";
                    %><option value="<%=options[i] %%" <%=selected
%>><%=options[i] %"></option><%
                }
                %>
            </select>

            <label for="pc-sl-pasteex">Paste Expiration:</label>
            <select id="pc-sl-pasteex" name="pcSlPasteex"
class="form-control">
                <%
                var options = [ "1 Day", "12 Hours", "3 Hours", "30
Minutes", "2 Minutes"];
                for ( var i = 0; i < options.length; i++ )
                {
                    var selected = ( therows[0].pasteex == options[i]
) ? "selected" : "";
                    %><option value="<%=options[i] %%" <%=selected
%>><%=options[i] %"></option><%
                }
                %>
            </select>

            <label for="pc-sl-pasteexpos">Paste Exposure:</label>
            <select id="pc-sl-pasteexpos" name="pcSlPasteexpos"
class="form-control">
                <%
                var options = [ "Public", "Unlisted" ];
                for ( var i = 0; i < options.length; i++ )

```

```

        {
            var selected = ( therows[0].pasteexpos ==
options[i] ) ? "selected" : "";
            %><option value="<%=options[i] %>" <%=selected
%>><%=options[i] %></option><%
        }
        %>
    </select>

    <label for="pc-tx-title">Title:</label>
    <input id="pc-tx-title" type="text" name="pcTxTitle"
class="form-control"
        value="<%= therows[0].title %>"
    <br />
    <input formaction="/manage/save-paste/<
%=therows[0].pastemanagerperm %>"
        id="pc-sb-savepaste" name="pcSbSavepaste"
class="btn btn-danger btn-block"
        type="submit" value="Save Paste" />
    <input formaction="/manage/delete-paste/<
%=therows[0].pastemanagerperm %>"
        id="pc-sb-delpaste" name="pcSbDelpaste"
class="btn btn-danger btn-block"
        type="submit" value="Delete Paste" />
    </div>
</div>
</div>
</div>

<div class="col-lg-6 col-md-6 col-sm-6 col-xs-6">
    <div class="panel panel-default">
        <div class="panel-heading">
            <h3 class="panel-title"><b>Links for This Paste</b></h3>
        </div>
        <div class="panel-body">
            <label for="pc-tx-managerurl">Manager URL:</label>
            <div id="pc-tx-managerurl" class="input-group">
                <input id="pc-in-tx-managerurl" type="text"
class="form-control"
                    value="<%=hostname%>/manage/<
%=therows[0].pastemanagerperm %>"
                <span class="input-group-btn">
                    <button id="pc-btn-copy-managerurl" class="btn

```

```
btn-default btn-copy" type="button"
                                data-clipboard-target="#pc-in-tx-
managerurl">Copy!</button>
        </span>
    </div>

    <label for="pc-tx-pasteurl">Paste URL:</label>
    <div id="pc-tx-pasteurl" class="input-group">
        <input id="pc-in-tx-pasteurl" type="text"
class="form-control"
                                value="<%=hostname%>/paste/<
%=therows[0].pasteperm
%>">
        <span class="input-group-btn">
            <button id="pc-btn-copy-pasteurl" class="btn btn-
default btn-copy" type="button"
                                data-clipboard-target="#pc-in-tx-
pasteurl">Copy!</button>
        </span>
    </div>
</div>
</div>
</div>
</div>
</div>
</form>
<nav class="navbar navbar-default navbar-inverse">
    <div class="container-fluid">
        <p class="navbar-text pull-right">Copyright &#169; Anonim 2017</p>
    </div>
</nav>
<script>
    new ClipboardJS('.btn-copy');
</script>
</body>

</html>
```

Perhatikan pada bagian:

```
<title>Manage <%= therows[0].title %> - Node.js Pasta</title>
```

Di sini kita menuliskan judul halaman dengan cara yang mirip hostname pada

halaman "Home".

Bedanya karena yang diinjeksi di server adalah row dari hasil query yang berupa array, maka di sana ada index 0 ("[0]").

Adapun ".title" di atas adalah nama kolom dari database.

Selanjutnya pada bagian ini:

```
<textarea id="pc-ta-newpaste" name="pcTaNewpaste" class="form-control" rows="10">
<%= therows[0].newpaste %>
</textarea>
```

Kita mengisi textarea paste yang berisi konten paste dengan nilai yang telah disimpan dari database.

Jadi ketika kita mengisi "abcde" pada halaman "Home", nilai itu akan dibawa ke halaman "Manage" di bagian textarea tersebut.

Lebih jelasnya, lihat kodenya di server (index.js):

```
app.get('/manage/:manageID', function (req, res) {
  db.all("SELECT * FROM pastes WHERE pastemanagerperm = '" + req.params.manageID + "'",
  function(err, rows) {
    if(rows.length > 0){
      res.render('manage', {
        therows : rows,
        hostname: getBaseUrl(req)
      });
    }else{
      res.status(404);
      res.render('404');
    }
  });
});
```

Di kode tersebut, kita meminta pada database untuk menampilkan row dari tabel pastes yang manage id-nya adalah "req.params.manageID".

Atau lebih tepatnya yang digarisbawahi ini, misalnya:

```
http://localhost:5000/manage/yanginilho
```

Cukup jelas bukan? ManagerID di database disimpan dalam kolom "pastemanagerperm".

Setelah query tersebut dijalankan, maka halaman "Manage" akan dirender di dalam callback.

Jika query tersebut hasilnya nihil, maka buka halaman 404.

Jadi di mana nilai "textarea" tadi?

Jawabannya adalah di dalam variabel "therows" yang diinjeksikan tadi.

Selanjutnya, bagian ini akan menampilkan URL manager yang digenerate:

```
<div id="pc-tx-managerurl" class="input-group">
<input id="pc-in-tx-managerurl" type="text" class="form-control" value="<%=hostname
%>/manage/<%=therows[0].pastemanagerperm %>">
<span class="input-group-btn">
<button id="pc-btn-copy-managerurl" class="btn btn-default"
type="button">Copy!</button>
</span>
</div>
```

Tepatnya bagian yang digarisbawahi tadi:

```
value="<%=hostname%>/manage/<%=therows[0].pastemanagerperm %>"
```

Hal yang sama berlaku juga pada URL paste yang di-generate.

URL tersebut bisa dicopy melalui sebuah button bernama "Copy" yang digarisbawahi tadi:

```
<button id="pc-btn-copy-managerurl" class="btn btn-default"
type="button">Copy!</button>
```

Lalu bagaimanakah fungsi copyToClipboard? Jawabnya adalah:

```
new ClipboardJS('.btn-copy');
```

Isi dari paste tersebut akan diupdate perubahannya atau dihapus melalui form ini:

```
<input formaction="/manage/save-paste/<%=therows[0].pastemanagerperm %>" id="pc-sb-
```

```

savepaste" name="pcSbSavepaste" class="btn btn-danger btn-block" type="submit"
value="Save Paste" />
<input formaction="/manage/delete-paste/<%=therows[0].pastemanagerperm %>" id="pc-sb-
delpaste" name="pcSbDelpaste" class="btn btn-danger btn-block" type="submit"
value="Delete Paste" />

```

Di sini kita tidak menggunakan tag "form" tapi "formaction", jadi dua tombol yang berbeda bisa memiliki action-nya masing masing, yakni yang digarisbawahi tadi.

Di sisi server (index.js), penge-save-an paste terjadi seperti ini:

```

app.post('/manage/save-paste/:manageID', function(req, res) {
  db.run(
    "UPDATE pastes SET newpaste=?,syntaxhi=?,pasteex=?,pasteexpos=?,title=? WHERE
    pastemanagerperm=?",
    req.body.pcTaNewpaste,
    req.body.pcSlSyntaxhi,
    req.body.pcSlPasteex,
    req.body.pcSlPasteexpos,
    req.body.pcTxTitle == "" ? "Untitled" : req.body.pcTxTitle,
    req.params.manageID
  );

  console.log("saved: " + req.params.manageID);

  setTimeout(function(){
    console.log("deleting the row automatically...");
    db.run("DELETE FROM pastes WHERE pastemanagerperm = '" + req.params.manageID +
    "'");
    console.log("deleted the row automatically...");
  }, textToMS(req.body.pcSlPasteex));

  res.redirect("/manage/" + req.params.manageID);
});

```

Perhatikan bahwa kita akan mengeksekusi query update ("UPDATE paste SET....").

Kemudian setelah di-update, kita akan menge-set timer kembali:

```

setTimeout(function(){
  console.log("deleting the row automatically...");
  db.run("DELETE FROM pastes WHERE pastemanagerperm = '" + req.params.manageID +

```

```
    "");  
    console.log("deleted the row automatically...");  
  }, textToMS(req.body.pcSlPasteex));
```

Dengan cara yang sama dengan bagian pembuatan database di bab sebelumnya.

Hal itu dilakukan karena di halaman "Manage" kita bisa meng-update Paste Expiration.

Selanjutnya penghapusan paste di sisi server (index.js):

```
app.post('/manage/delete-paste/:manageID', function(req, res) {  
  db.run("DELETE FROM pastes WHERE pastemanagerperm = '" + req.params.manageID +  
    "'");  
  console.log("deleted: " + req.params.manageID);  
  res.redirect("/");  
});
```

Tidak terlalu rumit. Kita hanya perlu melakukan query ("DELETE FROM pastes".

Kemudian, dilanjutkan dengan redirect ke halaman "Home" (res.redirect("/");).

Membuat Halaman Paste

Kita telah melewati bagian tersulit dari aplikasi ini.

Saatnya bersantai...

Sekarang buka file "paste.ejs" dari folder "ls-node-js-pasta/views".

Isinya kira-kira seperti ini:

```
<!DOCTYPE html>  
<html lang="en">  
  
<head>  
  <meta charset="utf-8">  
  <meta http-equiv="X-UA-Compatible" content="IE=edge">  
  <meta name="viewport" content="width=device-width, initial-scale=1">  
  
  <title><%= therows[0].title %> - Node.js Pasta</title>
```

```

    <meta name="description" content="Share <%= therows[0].title %> with Node.js
Pasta">
    <meta name="author" content="Anonim">

    <!-- [ Social Media meta tag ] -->
    <meta content='summary' name='twitter:card' />
    <meta content='@ciyy_space' name='twitter:site' />
    <meta content='@ciyy_space' name='twitter:creator' />

    <meta content='<%=hostname%>/paste/<%=therows[0].pasteperm %>'
property='og:url' />
    <meta content='<%= therows[0].title %> - Node.js Pasta' property='og:site_name' /
>
    <meta content='<%= therows[0].title %> - Node.js Pasta' property='og:title' />
    <meta content='<%= therows[0].title %> - Node.js Pasta' name='twitter:title' />
    <meta content='article' property='og:type' />
    <meta content='<%=hostname%>/views/img/ciyy-200x200.png' property='og:image' />
    <meta content='<%=hostname%>/views/img/ciyy-200x200.png' name='twitter:image' />
    <meta content='Share <%= therows[0].title %> with Node.js Pasta'
property='og:description' />
    <meta content='Share <%= therows[0].title %> with Node.js Pasta'
name='twitter:description' />
    <meta content='en_US' property='og:locale' />
    <!-- [/ Social Media meta tag ] -->

    <link rel="shortcut icon" type="image/png" href="/views/img/favicon.png" />

    <!-- Latest compiled and minified CSS -->
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/
bootstrap.min.css"
        integrity="sha384-BVYiISIFeK1dGmJRAkycuHAHRg320mUcww7on3RYdg4Va+PmSTsz/
K68vbdEjh4u" crossorigin="anonymous">

    <!-- Optional theme -->
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/
bootstrap-theme.min.css"
        integrity="sha384-
rHyoN1iRsVXV4nD0JutlNGaslCJuC7uwjduW9SVrLvRYooPp2bWYgmgJQIXw1/Sp"
crossorigin="anonymous">

    <link href="http://maxcdn.bootstrapcdn.com/font-awesome/4.1.0/css/font-
awesome.min.css" rel="stylesheet">

```



```

<script src="https://code.jquery.com/jquery-3.3.1.min.js"
    integrity="sha256-FgpCb/KJQlLNfOu91ta32o/NMZxltwRo8QtmkMRdAu8="
crossorigin="anonymous"></script>

<!-- Latest compiled and minified JavaScript -->
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"
    integrity="sha384-
Tc5IQib027qvyjSMfHj0MaLkfuWVxZxUPnCJA7l2mCWNIpG9mGCD8wGNlIcPD7Txa"
crossorigin="anonymous">
</script>

<script src="https://cdn.jsdelivr.net/npm/clipboard@2/dist/clipboard.min.js"></
script>
<!--
    <link href="/views/css/bootstrap.min.css" rel="stylesheet">
    <link href="/views/css/bootstrap-theme.min.css" rel="stylesheet">
    <link href="/views/css/font-awesome.min.css" rel="stylesheet">
    <script src="/views/js/jquery-3.2.1.min.js"></script>
    <script src="/views/js/bootstrap.min.js"></script>
-->

<style type="text/css" media="screen">
    #editor {
        position: relative;
        height: 300px;
    }
</style>
</head>

<body>

    <div class="container">
        <nav class="navbar navbar-default navbar-inverse">
            <div class="container-fluid">
                <div class="navbar-header">
                    <button type="button" class="navbar-toggle collapsed" data-
toggle="collapse" data-target="#navbar"
                        aria-expanded="false" aria-controls="navbar">
                        <span class="sr-only">Toggle navigation</span>
                        <span class="icon-bar"></span>
                        <span class="icon-bar"></span>
                        <span class="icon-bar"></span>
                    </button>

```

```

        <a class="navbar-brand" href="/">Node.js Pasta</a>
    </div>
    <div id="navbar" class="navbar-collapse collapse">
        <ul class="nav navbar-nav navbar-right">
            <li><a href="/trends">Trends</a></li>
            <li>
                <div class="btn-nav"><a class="btn btn-primary btn-small
navbar-btn" href="/"><span
                                class="glyphicon glyphicon-plus-sign"></span>
New Paste</a>
                </li>
            </ul>
        </div>
    </div>
</nav>

<div class="row">
    <div class="col-lg-12 col-md-12 col-sm-12 col-xs-12">
        <div class="panel panel-default">
            <div class="panel-heading">
                <h3 class="panel-title"><b>This Paste</b></h3>
            </div>
            <div class="panel-body">
                <div id="editor">
                    <%= therows[0].newpaste %>
                </div>
            </div>
        </div>
    </div>
</div>

<div class="row">
    <div class="col-lg-12 col-md-12 col-sm-12 col-xs-12">
        <div class="panel panel-default">
            <div class="panel-heading">
                <h3 class="panel-title"><b>Links for This Paste</b></h3>
            </div>
            <div class="panel-body">
                <label for="pc-tx-pasteurl">Paste URL:</label>
                <div id="pc-tx-pasteurl" class="input-group">
                    <input id="pc-in-tx-pasteurl" type="text" class="form-
control"
                                value="<%=hostname%>/paste/<%=therows[0].pasteperm

```

```
%>">
        <span class="input-group-btn">
            <button id="pc-btn-copy-pasteurl" class="btn btn-
default" type="button"
                data-clipboard-target="#pc-in-tx-pasteurl">Copy!
</button>
        </span>
    </div>
</div>
</div>
</div>
</div>
</div>
<div class="row">
    <div class="col-lg-12 col-md-12 col-sm-12 col-xs-12">
        <div class="panel panel-default">
            <div class="panel-heading">
                <h3 class="panel-title"><b>Share</b></h3>
            </div>
            <div class="panel-body">
                <div style="text-align: center">
                    <a href="http://www.facebook.com/share.php?u=<%=hostname
%>/paste/<%=therows[0].pasteperm %>&t=<%=therows[0].title %>"
                        class="fa fa-facebook fa-5x" target="_blank"></a>
                    <a href="http://twitter.com/share?original_referer=<
%=hostname%>/paste/<%=therows[0].pasteperm %>&source=web&text=<%=therows[0].title %>"
                        class="fa fa-twitter fa-5x" target="_blank"></a>
                    <a href="https://plus.google.com/share?url=<%=hostname%>/
paste/<%=therows[0].pasteperm %>"
                        class="fa fa-google fa-5x" target="_blank"></a>
                </div>
            </div>
        </div>
    </div>
</div>
</div>
</div>
<nav class="navbar navbar-default navbar-inverse">
    <div class="container-fluid">
        <p class="navbar-text pull-right">Copyright &#169; Anonim 2017</p>
    </div>
</nav>

<script src="/views/js/ace/ace.js" type="text/javascript"
```

```
charset="utf-8"></script>
  <script>
    var editor = ace.edit("editor");
    editor.setTheme("ace/theme/github");
    editor.getSession().setMode("ace/mode/<%=therows[0].syntaxhi %>");
    editor.setReadOnly(true);
  </script>
  <script>
    new ClipboardJS('#pc-btn-copy-pasteurl');
  </script>
</body>

</html>
```

Tidak ada yang rumit di sini, fungsi `copyToClipboard` yang sudah dijelaskan juga ada di sini.

Injeksi `therows` ke dalam file `"paste.ejs"` yang sudah dijelaskan juga ada.

Berarti tinggal dua hal lagi yang belum dijelaskan:

1. Membuat twitter card dan facebook open graph.
2. Memasang code editor read-only dengan Ace.

Twitter card adalah semacam screenshot yang ditampilkan saat kita men-share sebuah halaman web ke social media Twitter.

Facebook open graph juga mirip dengan itu.

Contohnya yang seperti ini:

Card preview



Twitter Card

Dan seperti ini di facebook:



Facebook Open Graph

Screenshot-screenshot tadi saya dapatkan dari debugger milik twitter dan

facebook.

Alamatnya di sini:

```
https://cards-dev.twitter.com/validator
```

```
https://developers.facebook.com/tools/debug/
```

Jadi, ketika Anda mempelajari tentang ini, lakukan percobaan dengan menggunakan website tersebut.

Dari sisi coding, yang menentukan tampilan tadi ada di bagian ini (ada di paste.ejs):

```
<!-- [ Social Media meta tag ] -->
<meta content='summary' name='twitter:card' />
<meta content='@ciyy_space' name='twitter:site' />
<meta content='@ciyy_space' name='twitter:creator' />

<meta content='<%=hostname%>/paste/<%=therows[0].pasteperm %>' property='og:url' />
<meta content='<%= therows[0].title %> - Node.js Pasta' property='og:site_name' />
<meta content='<%= therows[0].title %> - Node.js Pasta' property='og:title' />
<meta content='<%= therows[0].title %> - Node.js Pasta' name='twitter:title' />
<meta content='article' property='og:type' />
<meta content='<%=hostname%>/views/img/ciyy-200x200.png' property='og:image' />
<meta content='<%=hostname%>/views/img/ciyy-200x200.png' name='twitter:image' />
<meta content='Share <%= therows[0].title %> with Node.js Pasta'
property='og:description' />
<meta content='Share <%= therows[0].title %> with Node.js Pasta'
name='twitter:description' />
<meta content='en_US' property='og:locale' />
<!-- [/ Social Media meta tag ] -->
```

Untuk twitter, yang perlu diperhatikan adalah:

```
<meta content='summary' name='twitter:card' />
<meta content='<%= therows[0].title %> - Node.js Pasta' name='twitter:title' />
<meta content='<%=hostname%>/views/img/ciyy-200x200.png' name='twitter:image' />
```

Bagian pertama menentukan apakah tampilannya besar atau kecil. Dalam hal ini kecil.

Bagian kedua menentukan bagaimana title-nya.

Bagian ketiga menentukan apa gambarnya.

Untuk gambar, paling aman ambil resolusi 200x200 pixel.

Sisanya cukup jelas, yang penting di bagian meta name ada teks:

```
name='twitter:blablabla'
```

Jadi kalau:

```
name='twitter:description'
```

Ya... berarti untuk deskripsi dari link yang di-share.

Untuk Facebook, yang perlu diperhatikan adalah:

```
<meta content='<%=hostname%>/paste/<%=therows[0].pasteperm %>' property='og:url' />
<meta content='<%=hostname%>/views/img/ciyy-200x200.png' property='og:image' />
```

Bagian pertama harus diisi URL dari paste.

Bagian kedua gambarnya untuk yang kecil 200x200 pixel.

Ciri khas dari Facebook adalah ada teks:

```
property='og:blablabla'
```

Bentuk lainnya bisa ditafsirkan sendiri.

Selain itu, perhatikan juga pada bagian ini:

```
<script>
    var editor = ace.edit("editor");
    editor.setTheme("ace/theme/github");
    editor.getSession().setMode("ace/mode/<%=therows[0].syntaxhi %>");
    editor.setReadOnly(true);
</script>
```

Kode di atas menginisiasi tampilan code editor di halaman "Paste".

Theme yang digunakan adalah github dan bahasa yang digunakan ditentukan pada bagian ini:

```
"ace/mode/<%=therows[0].syntaxhi %>"
```

Code editor dibuat read-only karena di halaman "Paste", kita tidak diizinkan untuk meng-edit:

```
editor.setReadOnly(true);
```

Membuat Halaman Trends

Saat ini, kita telah menyentuh pembahasan terakhir.

Trends berguna untuk menampilkan paste dengan hits terbanyak.

Diurutkan dari yang terbanyak ke yang tersedikit dari atas ke bawah.

Pertama-tama, buka file "trends.ejs" dari folder "ls-node-js-pasta/views".

Isinya kira-kira begini:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1">

<title>Trends - Node.js Pasta</title>

<meta name="description" content="Node.js Pasta Trends!">
<meta name="author" content="Anonim">

<!-- [ Social Media meta tag ] -->
<meta content='summary' name='twitter:card' />
<meta content='@ciyy_space' name='twitter:site' />
<meta content='@ciyy_space' name='twitter:creator' />

<meta content='<%=hostname%>/trends' property='og:url' />
<meta content='Trends - Node.js Pasta' property='og:site_name' />
<meta content='Trends - Node.js Pasta' property='og:title' />
<meta content='Trends - Node.js Pasta' name='twitter:title' />
<meta content='website' property='og:type' />
<meta content='<%=hostname%>/views/img/ciyy-200x200.png' property='og:image' />
```



```

<meta content='<%=hostname%>/views/img/ciyy-200x200.png' name='twitter:image' />
<meta content='Node.js Pasta Trends!' property='og:description' />
<meta content='Node.js Pasta Trends!' name='twitter:description' />
<meta content='en_US' property='og:locale' />
<!-- [ / Social Media meta tag ] -->

<link rel="shortcut icon" type="image/png" href="/views/img/favicon.png" />
<link href="/views/css/bootstrap.min.css" rel="stylesheet">
<link href="/views/css/bootstrap-theme.min.css" rel="stylesheet">
<script src="/views/js/jquery-3.2.1.min.js"></script>
<script src="/views/js/bootstrap.min.js"></script>
</head>
<body>
<% include ./functions %>
<div class="container">
<nav class="navbar navbar-default navbar-inverse">
<div class="container-fluid">
<div class="navbar-header">
<button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-
target="#navbar" aria-expanded="false" aria-controls="navbar">
<span class="sr-only">Toggle navigation</span>
<span class="icon-bar"></span>
<span class="icon-bar"></span>
<span class="icon-bar"></span>
</button>
<a class="navbar-brand" href="/">Node.js Pasta</a>
</div>
<div id="navbar" class="navbar-collapse collapse">
<ul class="nav navbar-nav navbar-right">
<li><a href="/trends">Trends</a></li>
<li><div class="btn-nav"><a class="btn btn-primary btn-small navbar-btn"
href="/"><span class="glyphicon glyphicon-plus-sign"></span> New Paste</a></li>
</ul>
</div>
</div>
</nav>

<div class="row">
<div class="col-lg-12 col-md-12 col-sm-12 col-xs-12">
<div class="panel panel-default">
<div class="panel-heading">
<h3 class="panel-title"><b>Trending Pastes</b></h3>

```

```

</div>
<div class="panel-body" style="min-height: 336px; max-height: 336px; overflow-y:
scroll;">
<div class="list-group">
<% if (locals.therows) { %>
<% therows.forEach(function(singleRow) { %>
<a href="<%= hostname %>/paste/<%= singleRow.pasteperm %>" target="_blank"
class="list-group-item">
<h4 class="list-group-item-heading"><%= singleRow.title %></h4>
<p class="list-group-item-text"><i>Expired in <%= singleRow.pasteex %>, <%=
singleRow.hits %> Hit(s)</i></p>
<p class="list-group-item-text"><%= getTheExcerpt(singleRow.newpaste) %></p>
</a>
<% }); %>
<% } %>
</div>
</div>
</div>
</div>
</div>

<nav class="navbar navbar-default navbar-inverse">
<div class="container-fluid">
<p class="navbar-text pull-right">Copyright &#169; Anonim 2017</p>
</div>
</nav>
</body>
</html>

```

Tidak ada hal baru di sini, kecuali bagian:

```

<div class="list-group">
  <% if (locals.therows) { %>
    <% therows.forEach(function(singleRow) { %>
      <a href="<%= hostname %>/paste/<%= singleRow.pasteperm %>" target="_blank"
class="list-group-item">
        <h4 class="list-group-item-heading"><%= singleRow.title %></h4>
        <p class="list-group-item-text"><i>Expired in <%= singleRow.pasteex %>, <
%= singleRow.hits %> Hit(s)</i></p>
        <p class="list-group-item-text"><%= getTheExcerpt(singleRow.newpaste) %></
p>
      </a>
    } %>
  } %>
</div>

```

```
<% }); %>
<% } %>
</div>
```

Perhatikan bagian:

```
<% if (locals.therows) { %>
```

Di sini saya melakukan pengecekan apakah rows diinjeksi atau tidak. Karena apa yang saya tulis di sisi server ini (index.js):

```
app.get('/trends', function (req, res) {
  db.all("SELECT * FROM pastes WHERE pasteexpos='Public' ORDER BY hits DESC LIMIT
  20", function(err, rows) {
    res.render('trends', {
      therows : rows,
      hostname: getBaseUrl(req)
    });
  });
});
```

Lihat di sana bahwa saya tidak melakukan pengecekan jumlah rows di server.

Bandingkan dengan sisi server halaman "Manage":

```
app.get('/manage/:manageID', function (req, res) {
  db.all("SELECT * FROM pastes WHERE pastemanagerperm = '" + req.params.manageID
  + "'", function(err, rows) {
    if(rows.length > 0){
      res.render('manage', {
        therows : rows,
        hostname: getBaseUrl(req)
      });
    }else{
      res.status(404);
      res.render('404');
    }
  });
});
```

```
})
```

Di sisi server halaman "Manage" ada "if(rows.length > 0".

Sedangkan di sisi server halaman "Trends" tidak ada dan langsung render saja.

Apabila kita tidak melakukan pengecekan sama sekali maka di halaman "Trends" akan error atau undefined.

Maka itu, saya melakukan pengecekan di dalam template "paste.ejs":

```
<% if (locals.therows) { %>
```

Selanjutnya saya melakukan loop terhadap isi dari array "therows" diwakili oleh variabel "singleRow":

```
<% therows.forEach(function(singleRow) { %>
```

Dan pada akhirnya saya mengambil excerpt dari singleRow:

```
<%= getTheExcerpt(singleRow.newpaste) %>
```

Fungsi getTheExcerpt ini telah kita definisikan pada bab sebelumnya.

Tapi itu tidak bisa digunakan tanpa ini:

```
<% include ./functions %>
```

Dengan kode tersebut, kita meng-include template "functions.ejs" agar fungsi getTheExcerpt dapat digunakan.

Mencoba Aplikasi Ini

Sekarang, kita telah selesai membahas pengerjaan aplikasi Node.js Pasta.

Mungkin ada satu atau beberapa kode yang tertinggal, Anda tinggal mengcopy-paste saja dari source code buku ini.

Atau boleh juga langsung jalankan source code buku ini. Yang penting Anda mengerti cara kerjanya.

Jadi... Jalankan perintah ini:

```
node .
```

Kemudian saksikan hasilnya di browser Anda.

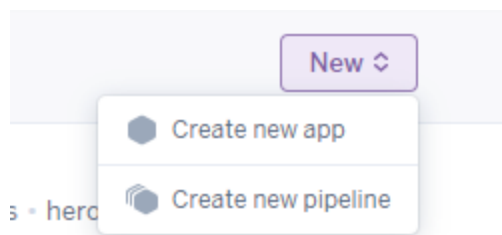
Men-Deploy Aplikasi Ini di Heroku

Untuk mencoba tampilan sharing di social media, kita membutuhkan sebuah hosting.

Yang kita bahas adalah heroku (<https://www.heroku.com>).

Daftarkan akun Anda terlebih dahulu.


Kemudian, buat app baru dan beri nama yang available:



Buat App Baru

App name

Choose a region

 United States

Create app

Isian Nama App

Setelah selesai, Anda siap untuk meng-uploadnya. Pastikan Anda sudah menginstall semua software yang dibutuhkan di awal Bab 13.

Sekarang buka command line, masuk ke folder "ls-node-js-pasta".

Jalankan perintah ini:

```
heroku login
```

Isi email dan password Anda.

Kemudian jalankan perintah ini:

```
git init  
heroku git:remote -a ls-node-js-pasta
```

```
git add .  
git commit -am "make it better"  
git push heroku master
```

Tunggu sampai proses upload selesai.

Setelah itu buka browser ke:

```
https://ls-node-js-pasta.herokuapp.com
```

Untuk mengupdate saja, cukup dengan perintah ini:

```
heroku login  
git add .  
git commit -am "make it better"  
git push heroku master
```

Ada kemungkinan proses tersebut gagal.

Alasannya, saya telah membuat subdomain bernama ls-node-js-pasta di Heroku.

Jadi solusinya adalah dengan mengganti nama "ls-node-js-pasta" dengan nama lain pilihan Anda.

Nama subdomain sama dengan nama App yang telah kita berikan.

Bab 14. Praktikum: Membuat Web System Monitor

Aplikasi berbasis web semakin mudah dibuat akhir-akhir ini.

Dengan lahirnya beberapa front-end javascript framework seperti Vue.js, pengembangan aplikasi web menjadi lebih ter-manage dengan baik.

Apalagi, front-end framework tersebut didukung oleh teknologi back-end dari Node.js.

Dengan menggunakan Node.js, kita dapat melakukan sebuah operasi secara nonblocking.

Artinya, server bisa merespon request lebih cepat.

Dengan adanya peluang tersebut, saya sebagai penulis berpendapat bahwa ada gunanya mempelajari teknologi-teknologi tersebut.

Oleh karena itulah, kali ini kita akan mempelajari pembuatan aplikasi web berbasis Node.js dan framework front-end yang cukup populer, yakni Vue.js.

Apa yang Akan Kita Buat

Kali ini, kita akan belajar membuat aplikasi web system monitoring dengan Node.js, Express.js, Vue.js, dan Chart.js.

Aplikasi ini berguna untuk memantau system performance, misalnya, CPU usage, memory usage, dan berbagai info lainnya.

Kurang lebih, aplikasi ini mirip Task Manager-nya Windows, tetapi berjalan di web.

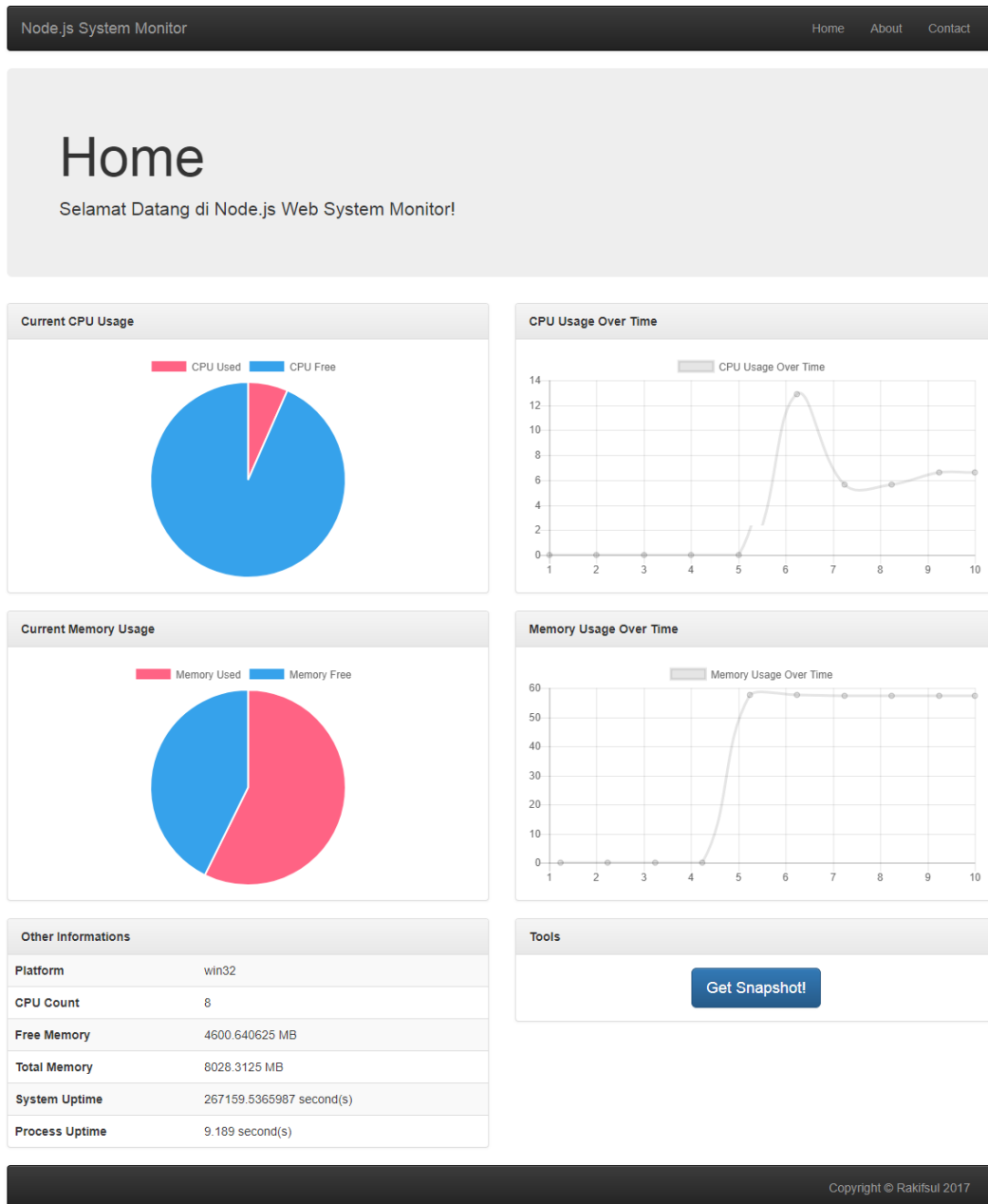
Akan tetapi, berbeda dengan Windows Task Manager, aplikasi ini akan mengukur system performance dari server di mana aplikasi ini di-host, bukan untuk komputer user, walaupun user dapat melihat hasilnya di komputer user melalui browser.

Adapun data-data yang akan diperoleh melalui aplikasi ini adalah:

- Current CPU usage
- CPU usage over time
- Current memory usage
- Memory usage over time

- Jumlah core CPU
- Platform server yang meng-host aplikasi ini
- Free memory
- Total memory
- System uptime
- Process uptime

Untuk melihat aplikasi ini yang sudah jadi, Anda boleh lihat screenshotnya di sini:



Node.js Web System Monitor

Mengapa Kita Membuatnya

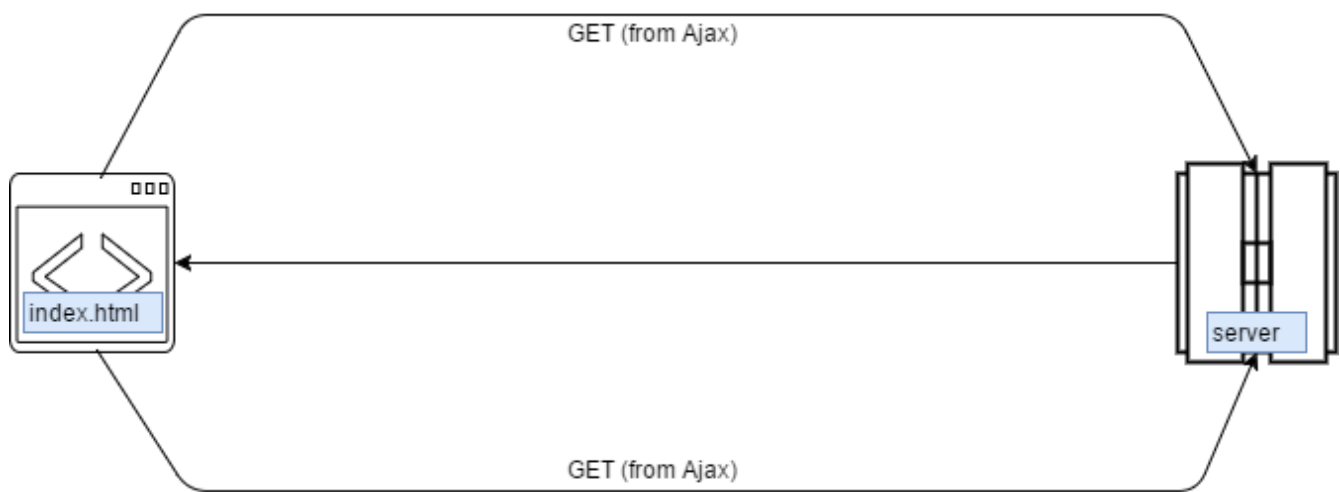
Aplikasi web system monitoring ini berguna untuk memantau server performance, sehingga kita dapat dengan mudah mengetahui apakah server sedang terbebani atau tidak.

Selain itu, pembuatan aplikasi ini sangat bermanfaat untuk mempelajari konsep-konsep yang berkaitan dengan Node.js dan package-package pendukungnya dan yang terpenting adalah memberikan gambaran umum bagaimana aplikasi web single-page berbasis Node.js dibuat.

Bagaimana Kita Membuatnya

Untuk membuatnya, kita perlu memahami rancangan dari aplikasi ini.

Perhatikanlah gambar di bawah ini.



Cara Kerja Aplikasi

Pertama-tama, kita berasumsi bahwa Node.js telah terinstall di server.

Begitu pula dengan script aplikasi ini yang telah selesai dibuat.

Pada saat user membuka URL (misalnya server Anda adalah localhost port 5000):

<http://localhost:5000/>

Maka server akan meresponnya dengan sebuah file "index.html" yang diolah di browser.

Setelah itu "index.html" akan mengeksekusi AJAX untuk melakukan request ke localhost port 5000 mengenai data yang harus ditampilkan di browser, misalnya CPU usage, memory usage dan lain lain.

Eksekusi javascript ini akan dilakukan setiap selang waktu satu detik.

Server akan meresponnya dengan data terkait dalam format JSON.

Data dalam bentuk JSON tersebut akan diolah di browser menjadi data yang siap saji, misalnya dalam bentuk grafik.

Selanjutnya javascript di "index.html" akan mengupdate grafik juga setiap selang waktu satu detik.

Hal ini terjadi terus-menerus sampai user menutup halaman web tersebut.

Yang Anda Butuhkan sebelum Kita Belajar

Perlu saya akui, bahwa level artikel ini adalah menengah. Saya berasumsi bahwa Anda telah memahami pemrograman javascript dan setidaknya pernah membuat aplikasi web sebelumnya.

Di samping itu, Anda juga membutuhkan ini:

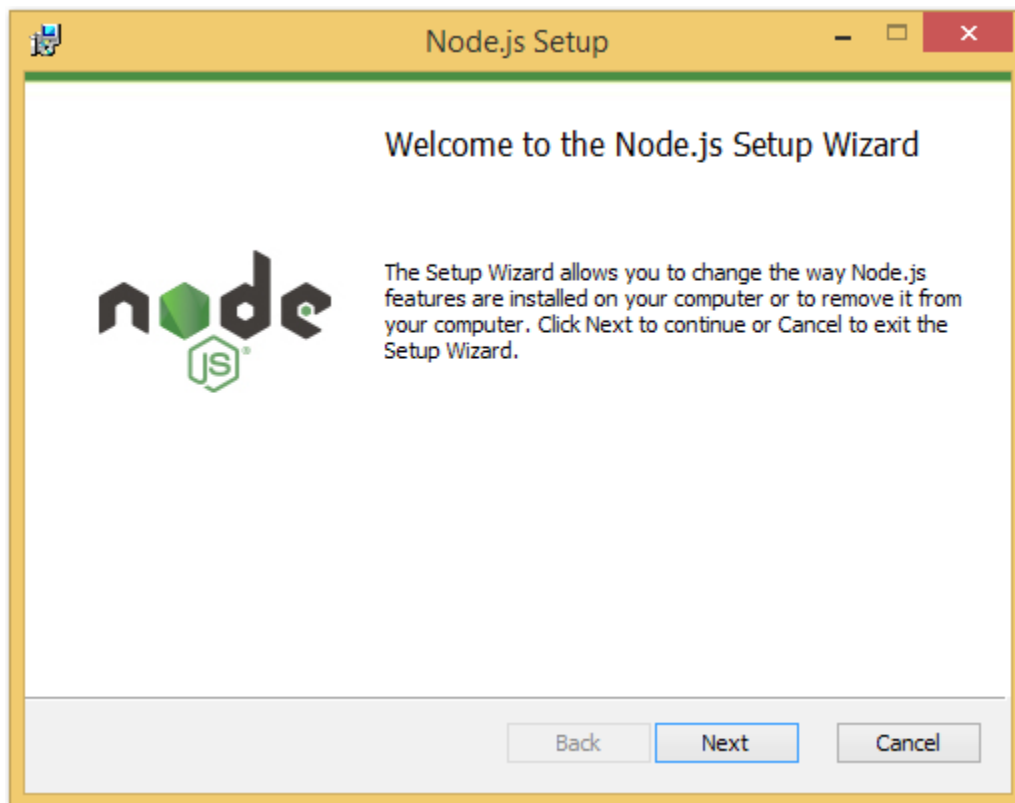
- Node.js dan NPM.
- Text Editor apapun, tapi saya menyarankan Visual Studio Code (gratis, silahkan cari di google).
- Web Browser apapun, tapi saya menyarankan Chrome.
- Koneksi internet.

Menginstall Node.js dan NPM

Langkah pertama dalam pembuatan system monitor ini adalah menginstall Node.js dan NPM. Caranya adalah dengan mendownload installer Node.js dari situs resminya.

Installer tersebut sudah mencakup Node.js dan NPM.

Cara menginstallnya cukup sederhana, hanya klik next dan finish saja.



Menginstall Node.js dan NPM

Setelah Node.js terinstall, maka NPM juga akan terinstall.

Membuat Project Node.js

Setelah menginstall Node.js dan NPM, langkah selanjutnya adalah membuat project Node.js.

Pertama-tama, buatlah folder dengan nama "Is-web-system-monitor".

Lokasi folder tersebut bisa di mana saja, pastikan di tempat yang mudah dijangkau.

Untuk membuatnya, bisa menggunakan Right Click>New>Folder di Windows Explorer.

Setelah itu, masuklah ke dalam folder tersebut.

Lalu di Windows Explorer, pergi ke menu File>Open Windows Power Shell>Open Windows Power Shell as Administrator.

Dengan mengklik menu tersebut, Power Shell akan dibuka sebagai administrator pada folder "ls-web-system-monitor".

Power Shell ini sebenarnya mirip dengan Command Prompt, hanya saja ada fitur tambahan seperti beberapa perintah yang mirip dengan Terminal yang ada di Linux.

Setelah Power Shell dibuka ketik:

```
npm init
```

Nanti akan diajukan beberapa pertanyaan. Isilah dengan jawaban ini:

```
name: ls-web-system-monitor
version: 1.0.0
description: terserah
entry point: server.js
test command: kosongkan
git repository: kosongkan
keywords: kosongkan
author: nama Anda
license: ISC
is this ok: yes
```

Pastikan Anda berada dalam folder "ls-web-system-monitor".

Maka project Node.js dengan nama "ls-web-system-monitor" selesai dibuat.

Menginstall Package yang Dibutuhkan

Setelah project Node.js dibuat, kita akan menginstall package yang diperlukan dalam project ini.

Ada cukup banyak package yang diperlukan, tetapi cara menginstallnya cukup mudah.

Vue.js

```
npm install vue --save
```

Vue.js adalah package Node.js yang berfungsi sebagai framework untuk front-end. Framework ini berguna untuk mengembangkan aplikasi web yang single-page seperti yang sedang kita buat ini.

Perhatikan bahwa di sini kita menggunakan argument `--save`. Tujuannya adalah agar package yang kita install akan dimasukkan ke dalam daftar package di "package.json". Dengan ini, apabila kita memiliki source code tanpa package, kita akan bisa menginstall semua package sekaligus dengan:

```
npm install
```

Vue-Resource

```
npm install vue-resource --save
```

Vue-Resource adalah HTTP client untuk Vue.js. Dengan ini kita bisa menggunakan HTTP request dari client side.

Bootstrap

```
npm install bootstrap --save
```

Bootstrap adalah library CSS untuk membuat tampilan web menjadi responsive.

JQuery

```
npm install jquery --save
```

JQuery dibutuhkan karena Bootstrap membutuhkannya.

Chart.js

```
npm install chart.js --save
```

Chart.js adalah package Node.js yang berfungsi sebagai library yang dapat menampilkan grafik. Kita akan membuat grafik pie dan line dengan Chart.js.

OS-Utills Module

```
npm install os-utils --save
```

Package ini berfungsi untuk mendapatkan informasi terkait dengan operating system seperti jumlah core pada CPU, penggunaan CPU, penggunaan memory, system uptime, dan process uptime.

Expressjs

```
npm install express --save
```

Package ini merupakan framework Node.js. Framework ini berfungsi untuk menyederhanakan routing dari aplikasi yang akan kita buat.

FS Module

Module ini berfungsi untuk menulis file snapshot aplikasi ini yang nantinya akan bisa di-download oleh user ke komputer mereka. Kita tidak akan menginstall module ini, karena module ini sudah ada secara default bersama Node.js.

Setelah package tersebut kita install, perhatikanlah pada folder "ls-web-system-monitor". Di sana akan kita jumpai folder baru bernama "node_modules". Di sinilah semua package yang kita install berada.

Mengimpor Semua Package yang Telah Kita Install

Setelah semua package kita install, kita harus mengimpornya terlebih dahulu sebelum dapat digunakan.

Pertama-tama buatlah file bernama "server.js" di dalam folder "ls-web-system-monitor".

Selanjutnya, untuk mengimpor salah satu package, buka file target lalu tulis:

```
var apapun = require("nama_package");
```

Jadi, jika kita akan mengimpor package "os-utils", kita akan menulis ini pada file "server.js":

```
var osu = require("os-utils");
```

Kali ini, kita akan mengimpor semua package yang telah kita install, maka tuliskan ini pada file "server.js":

```
var osu = require('os-utils');  
var fs = require('fs');  
var express = require('express');  
var app = express();
```

Perhatikan bahwa pada saat mengimport Express.js, ada lanjutan seperti ini:

```
var app = express();
```

Hal itu dilakukan karena pada saat kita me-require Express.js, yang diimpor adalah fungsi. Untuk mendapatkan objectnya, fungsi harus dijalankan.

Lalu bagaimana dengan Vue.js dan Chart.js?

Kedua package tersebut merupakan package front-end. Dengan kata lain, package tersebut hanya digunakan dalam file HTML yang akan dieksekusi dari client.

Sementara, file "server.js" adalah server side script yang berjalan di server. Jadi file "server.js" tidak membutuhkan kedua package tersebut untuk dijalankan di server.

Untuk mengimpor kedua package tersebut, kita harus membuat file HTML terlebih dahulu, lalu mengimpornya seperti mengimpor javascript dalam HTML.

Hal itu akan dibahas pada bagian selanjutnya.

Sementara ini, fokuskan perhatian Anda terlebih dahulu pada server.

Membuat Kerangka Request Handler

Mengetes Request Handler Express

Sampai saat ini, secara teori kita sudah bisa membuat HTTP server sederhana.

Untuk mencobanya, tulis kode di bawah ini pada file baru bernama "servertest.js":

```
var express = require('express');
var app = express();

app.get('/', function (req, res) {
  res.writeHead(200, {"Content-Type": "text/plain"});
  res.end("Hello World\n");
});

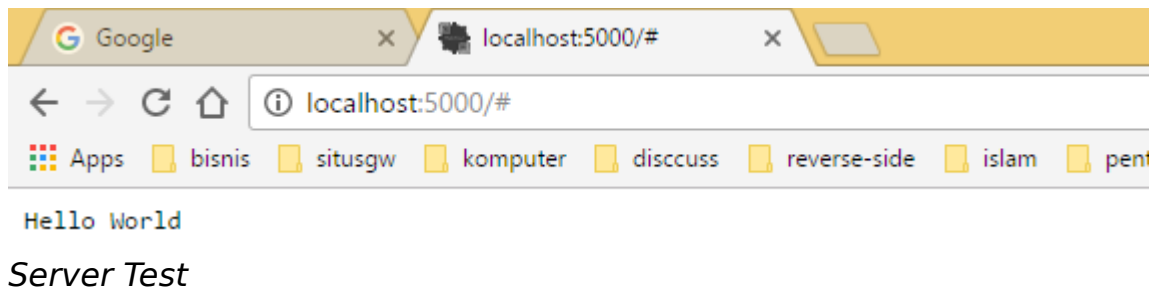
app.listen(5000, function () {
  console.log('Node app is running on port 5000');
});
```

Saya sengaja menuliskannya pada file lain agar konsentrasi kita pada file "server.js" tidak buyar.

Setelah menuliskannya, jalankan perintah ini pada Power Shell:

```
node servertest.js
```

Tampilannya akan seperti ini:



Penjelasannya...

Pada saat user memasukkan input "http://localhost:5000/" pada browser, yang artinya browser akan menyambungkan dirinya dengan server pada alamat localhost, port 5000, maka request "/" akan diminta pada "servertest.js".

Request ini disaring melalui script "servertest.js" pada baris kode:

```
app.get('/', function (req, res) { //YANG INI
  res.writeHead(200, {"Content-Type": "text/plain"});
  res.end("Hello World\n");
});
```

Tentu saja itu hanya bisa terjadi apabila "servertest.js" memiliki instruksi ini:

```
app.listen(5000, function () { //YANG INI
  console.log('Node app is running on port 5000');
});
```

Jadi fungsi get akan menyaring request GET dari browser, dan fungsi "app.listen" akan mendengarkannya pada port yang diberikan, dalam hal ini 5000.

Mengembangkan Request Handler Express Pada Aplikasi Kita

Dengan dasar percobaan tadi, sekarang kita bisa mengembangkan request handler untuk menangani semua request yang relevan pada aplikasi yang sedang kita kerjakan ini.

Anda boleh menghapus atau memindahkan file "servertest.js" ke tempat lain agar tidak mengganggu konsentrasi. Itu terserah Anda.

Yang penting, sekarang kita fokuskan kembali pikiran kita pada file "server.js".

Sekarang, kita akan menambahkan baris kode ini di file "server.js":

```
app.get('/', function (req, res) {
  //di sini kita akan mengembalikan file index.html
});
app.get('/api/home', function (req, res) {
  //di sini kita akan mengembalikan json untuk konten home
});
app.get('/api/about', function (req, res) {
  //di sini kita akan mengembalikan json untuk konten about
});
app.get('/api/contact', function (req, res) {
  //di sini kita akan mengembalikan json untuk konten contact
});
app.get('/api/os_utils_cpu_usage', function (req, res) {
  //di sini kita akan mengembalikan json untuk nilai cpu usage
});
app.get('/api/os_utils_memory_usage', function (req, res) {
  //di sini kita akan mengembalikan json untuk nilai memory usage
});
app.get('/api/os_utils_others', function (req, res) {
  //di sini kita akan mengembalikan json untuk info lainnya
});
app.get('/api/download_snapshot', function (req, res) {
  //di sini kita akan mengembalikan file snapshot untuk didownload
});
```

Tujuan dari setiap request handler dijelaskan melalui komentarnya. Semua dapat dijelaskan melalui komentar tersebut.

Hanya saja, perhatikan pada:

```
app.get('/', function (req, res) {
  //di sini kita akan mengembalikan file index.html
});
```

dan

```
app.get('/api/download_snapshot', function (req, res) {  
  //di sini kita akan mengembalikan file snapshot untuk didownload  
});
```

Yang pertama, "app.get('/'".

Di sana kita mengembalikan response berupa file HTML.

Sedangkan yang ke-2, "app.get('/api/download_snapshot' ...".

Di sana kita akan mengembalikan file snapshot untuk didownload.

Dengan kata lain, yang ke-2 adalah download link.

Sisanya, semua request diresponse dengan JSON.

Ini adalah prinsip utama dalam single-page web application, di mana hanya sebuah file HTML saja yang dikembalikan, sedangkan serah-terima data lainnya dari aplikasi tersebut dilakukan melalui JSON yang diterima melalui AJAX.

Dengan cara tersebut, user tidak perlu me-refresh halaman untuk mendapatkan update.

Jadi, sampai saat ini keseluruhan kode dalam file "server.js" adalah:

```
var osu = require('os-utils');  
var fs = require('fs');  
var express = require('express');  
var app = express();  
  
app.get('/', function (req, res) {  
  //di sini kita akan mengembalikan file index.html  
});  
app.get('/api/home', function (req, res) {  
  //di sini kita akan mengembalikan json untuk konten home  
});  
app.get('/api/about', function (req, res) {  
  //di sini kita akan mengembalikan json untuk konten about  
});  
app.get('/api/contact', function (req, res) {  
  //di sini kita akan mengembalikan json untuk konten contact  
});
```

```
app.get('/api/os_utils_cpu_usage', function (req, res) {
  //di sini kita akan mengembalikan json untuk nilai cpu usage
});
app.get('/api/os_utils_memory_usage', function (req, res) {
  //di sini kita akan mengembalikan json untuk nilai memory usage
});
app.get('/api/os_utils_others', function (req, res) {
  //di sini kita akan mengembalikan json untuk info lainnya
});
app.get('/api/download_snapshot', function (req, res) {
  //di sini kita akan mengembalikan file snapshot untuk didownload
});

app.listen(5000, function () {
  console.log('Node app is running on port', 5000);
});
```

Anda bisa mengetest server dengan:

```
node server.js
```

Sampai saat ini server belum bisa memberikan output apapun, tetapi telah melakukan listen di port 5000.

Membuat Pilihan Port yang Didengar Lebih Dinamis

Walaupun sebenarnya cukup dengan hanya membiarkan port yang didengarkan adalah port 5000, ada cara agar port yang kita tentukan lebih dinamis.

Jadi kita dapat mengganti port yang didengar aplikasi ini dengan port lain tanpa mengubah kode.

Caranya dengan menggunakan:

```
process.env.PORT
```

Dengan menggunakan itu, maka kita bisa menge-set port dalam environment variable, lalu nilai port tersebut akan digunakan dalam kode kita.

Sekarang coba tulis kode ini pada file "server.js" di atas request handler dan di bawah require:

```
//...
var express = require('express');
var app = express();
app.set('port', (process.env.PORT || 5000)); //TAMBAHKAN INI

//request handlers...
app.get('/')...
```

Maka sampai saat ini, kode file "server.js" adalah seperti ini:

```
var osu = require('os-utils');
var fs = require('fs');
var express = require('express');
var app = express();

app.set('port', (process.env.PORT || 5000)); //ADA TAMBAHAN INI

app.get('/', function (req, res) {
    //di sini kita akan mengembalikan file index.html
});
app.get('/api/home', function (req, res) {
    //di sini kita akan mengembalikan json untuk konten home
});
app.get('/api/about', function (req, res) {
    //di sini kita akan mengembalikan json untuk konten about
});
app.get('/api/contact', function (req, res) {
    //di sini kita akan mengembalikan json untuk konten contact
});
app.get('/api/os_utils_cpu_usage', function (req, res) {
    //di sini kita akan mengembalikan json untuk nilai cpu usage
});
app.get('/api/os_utils_memory_usage', function (req, res) {
    //di sini kita akan mengembalikan json untuk nilai memory usage
});
app.get('/api/os_utils_others', function (req, res) {
    //di sini kita akan mengembalikan json untuk info lainnya
});
app.get('/api/download_snapshot', function (req, res) {
    //di sini kita akan mengembalikan file snapshot untuk didownload
});
```

```
//ADA PERUBAHAN INI
app.listen(app.get('port'), function () {
  console.log('Node app is running on port', app.get('port'));
});
```

Lalu, pada Power Shell, ketikkan:

```
$env:PORT = 1234
```

Kemudian jalankan:

```
node server.js
```

Maka aplikasi ini akan memberitahu:

```
Node app is running on port 1234
```

Jika pada Power Shell kita langsung menjalankan aplikasi ini tanpa:

```
$env:PORT = 1234
```

Maka dia akan berjalan pada port 5000:

```
Node app is running on port 5000 (*)
```

Catatan:

(*)Untuk memastikannya, tutup Power Shell, lalu buka kembali pada folder "Is-web-system-monitor" sebagai administrator, lalu jalankan kembali aplikasi ini.

Merancang User Interface

Sekarang saatnya merancang user interface. Untuk melakukannya ada beberapa hal yang perlu dilakukan di server, dan ada pula beberapa hal yang perlu dilakukan di client.

Memberi Akses File Statis Dari Server

Agar user dapat men-download single-page HTML yang akan menampilkan user interface beserta favicon dan client side scriptnya, kita perlu mengonfigurasi Express.js terlebih dahulu.

Dengan demikian, setelah hal tersebut dilakukan, user akan bisa mengakses

favicon dalam file HTML dengan cara seperti ini:

```
<link rel="shortcut icon" type="image/png" href="assets/images/favicon.png"/>
```

atau memasukkan javascript dari file HTML dengan cara seperti ini:

```
<script src="scripts/jquery.min.js"></script>
<script src="scripts/js/bootstrap.min.js"></script>
<script src="scripts/vue.js"></script>
<script src="scripts/vue-resource.min.js"></script>
<script src="scripts/Chart.js"></script>
```

atau memasukkan script CSS dari file HTML dengan cara seperti ini:

```
<link rel="stylesheet" href="scripts/css/bootstrap.min.css">
<link rel="stylesheet" href="scripts/css/bootstrap-theme.min.css">
<link rel="stylesheet" href="assets/css/style.css">
```

Perhatikan bagian yang digaris bawah. Secara default, Node.js tidak mengizinkan hal seperti ini.

Dengan kata lain, melakukan request misalnya:

```
http://localhost:5000/scripts/vue.js
```

Tidak akan dianggap sebagai request yang sah, karena tidak ada request handler untuk hal tersebut (Anda bisa cek lagi request handler yang telah kita buat sebelumnya).

Kecuali jika kita meminta izin terlebih dahulu kepada server di file "server.js".

Untuk meminta izin kepada server agar server bisa menyediakan akses kepada file-file tadi, tulis kode di bawah ini di file "server.js":

```
//TAMBAHKAN INI
app.use('/scripts', express.static(__dirname + '/node_modules/chart.js/dist/'));
app.use('/scripts', express.static(__dirname + '/node_modules/vue/dist/'));
app.use('/scripts', express.static(__dirname + '/node_modules/vueresource/
dist/'));
app.use('/scripts', express.static(__dirname + '/node_modules/bootstrap/dist/'));
app.use('/scripts', express.static(__dirname + '/node_modules/jquery/dist/'));
app.use('/assets', express.static(__dirname + '/assets/'));
```

```
////////////////////////////////////  
app.set('port', (process.env.PORT || 5000)); // DI ATAS INI  
//....
```

Dengan cara ini, maka folder "__dirname/node_modules/chart.js/dist/" akan dipetakan di URL sebagai:

```
http://localhost:5000/scripts/
```

Jadi, jika kita merequest:

```
http://localhost:5000/scripts/vue.js
```

Maka request itu menjadi sah dan browser akan menerima script "vue.js" sebagai response.

Hal yang sama berlaku dengan "assets". Bedanya kita akan menempatkan file "index.html", "favicon.png", "style.css", dan file snapshot di sana.

Saya sengaja membedakan request untuk package front-end yang telah kita install dengan NPM dengan file statis buatan kita di tempat berbeda agar kita tidak bingung.

Selain itu, perhatikan juga bahwa kita menggunakan variabel global "__dirname".

Variabel ini akan mengembalikan nilai string dari direktori di mana file "server.js" berada.

Jadi kita memberikan informasi pada server tentang lokasi file statis kita dengan menggabungkan "__dirname" dengan direktori di mana file target berada.

Mencoba Mengakses File Statis dari Server

Untuk memastikan bahwa cara ini berhasil, kita akan membuat response bagi request "/" berupa file "index.html".

Pertama-tama, buatlah sebuah folder bernama "assets" di dalam folder "ls-web-system-monitor".

Caranya dengan menggunakan Right Click>New>Folder. Pastikan bahwa folder "assets" berada di dalam folder "ls-web-system-monitor".

Lalu, buatlah folder bernama "css" di dalam folder "assets".

Kemudian, buatlah file baru bernama "index.html" di dalam folder "assets" dan file baru bernama "style.css" di dalam folder "css".

Isi dari file "style.css":

```
/*Ini Adalah File Style.css*/
```

Sedangkan isi dari file "index.html":

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>Node.js System Information - App By Rakifsul</title>
<link rel="stylesheet" href="assets/css/style.css">
</head>
<body>
INI ADALAH FILE INDEX.HTML
</body>
</html>
```

Selain itu, kita memerlukan perubahan pada request handler "/" yang telah kita buat:

```
app.get('/', function (req, res) {
  res.sendFile("assets/index.html");
});
```

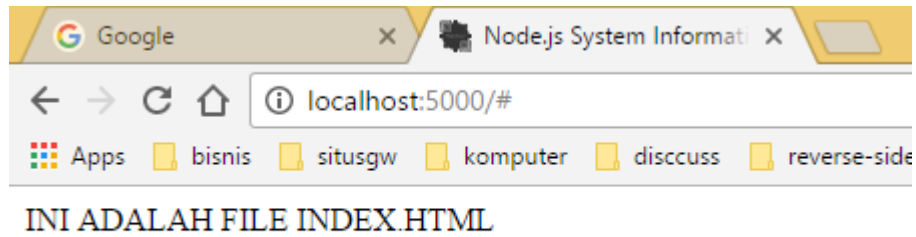
Setelah itu, jalankan aplikasi dengan:

```
node server.js
```

Maka dengan melakukan request melalui browser:

```
http://localhost:5000/
```

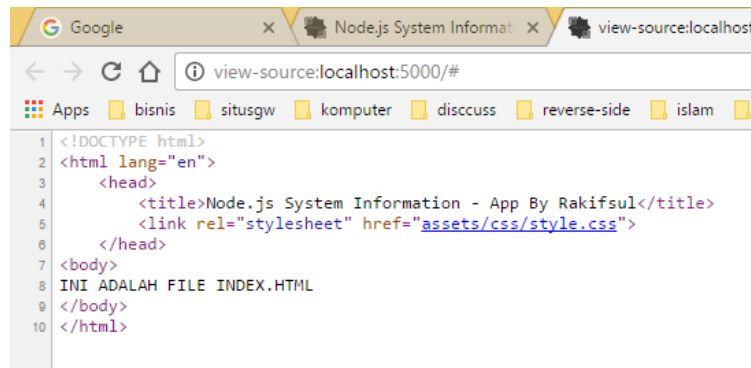
Akan tampil halaman web seperti ini:



Tampilan index.html

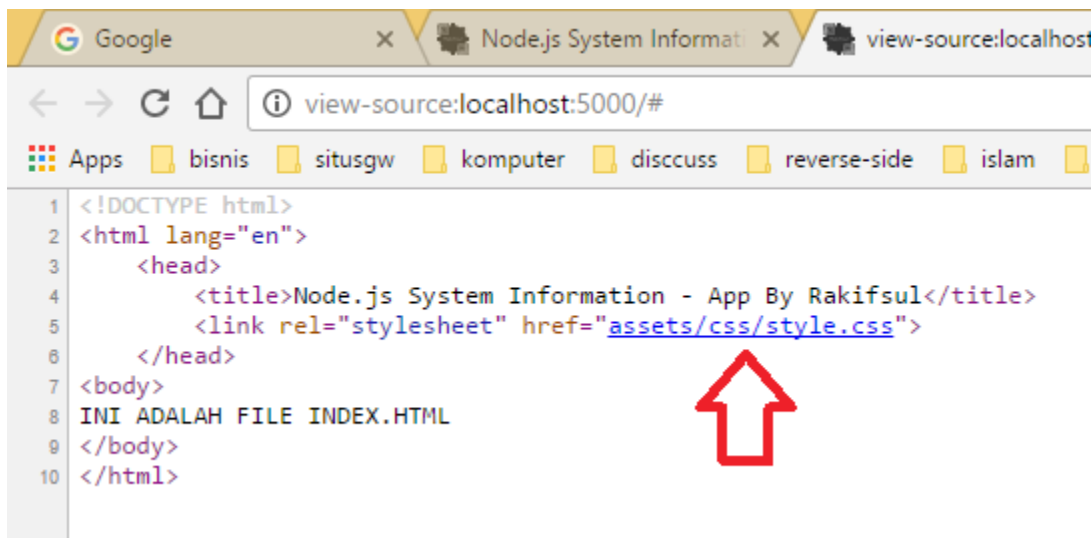
Mari kita gali lebih lanjut dengan Right Click>View page source.

Nanti source dari halaman tersebut akan dibuka:



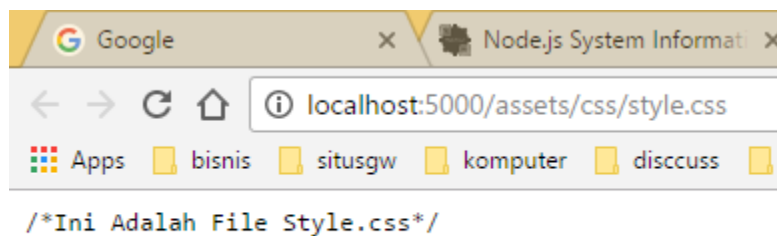
Source index.html

Selanjutnya cobalah klik bagian sini:



Click Bagian Ini

Ternyata, file "style.css" dapat diterima, yang artinya request tersebut adalah sah:



CSSnya

Merespon Setiap Request dalam Bentuk JSON dari Server

Setelah kita membuktikan bahwa izin untuk mengakses file statis dari server berjalan semestinya, sekarang saatnya kita membuat response dari setiap request yang telah kita definisikan.

Pada tahap ini, kita akan mengisi tiap request handler yang kita buat dengan kode-kode yang relevan di file "server.js".

Hanya pada request "/" saja kita akan membiarkan kodenya seperti sebelumnya, yakni:

```
app.get('/', function (req, res) {  
  res.sendFile("assets/index.html");  
});
```

Karena bagian itu tidak perlu diubah. Perubahan hanya akan dilakukan pada file "index.html" yang akan dilakukan nanti.

Response untuk Home, About, dan Contact

Untuk request "api/home", "api/about", dan "api/contact" kita hanya akan mengembalikan response berupa teks statis:

```
app.get('/api/home', function (req, res) {  
  res.json({title: "Home", message: "Selamat Datang di Node.js Web System Monitor!"});  
});  
  
app.get('/api/about', function (req, res) {  
  res.json({title: "About", message: "Node.js Web System Monitor dibuat oleh saya."});  
});  
  
app.get('/api/contact', function (req, res) {  
  res.json({title: "Contact", message: "Kontak Saya di http://www.ciyv.space atau Email  
ke rakifsul@gmail.com"});  
});
```

Response tersebut akan diberikan ketika user mengakses navigasi "Home", "About", dan "Contact".

Karena tema kita kali ini adalah single page application, maka navigasi tersebut di-respond dengan JSON melalui AJAX.

Response untuk System Performance

Selanjutnya, kita akan membuat response mengenai system performance:

```
var cpuUsageVal = 0;  
app.get('/api/os_utils_cpu_usage', function (req, res) {  
  osu.cpuUsage(function(v){  
    cpuUsageVal = v;  
  });  
});
```

```

    res.json({name: "CPU Usage", value: cpuUsageVal});
  });

app.get('/api/os_utils_memory_usage', function (req, res) {
  res.json({name: "Memory Usage", value: osu.freememPercentage()});
});

app.get('/api/os_utils_others', function (req, res) {
  var ret = {
    platform: osu.platform(),
    cpuCount: osu.cpuCount(),
    freeMem: osu.freemem(),
    totalMem: osu.totalmem(),
    sysUpTime: osu.sysUptime(),
    processUpTime: osu.processUptime()
  };
  res.json({name: "Other Informations", value: ret});
});

```

Tidak ada yang membingungkan di kode tersebut, semua kode dapat dipahami dari sintaksnya.

Kecuali bagian:

```

var cpuUsageVal = 0;
app.get('/api/os_utils_cpu_usage', function (req, res) {
  osu.cpuUsage(function(v){
    cpuUsageVal = v;
  });
  res.json({name: "CPU Usage", value: cpuUsageVal});
});

```

Perhatikan bahwa di sini kita mendeklarasikan sebuah variabel bernama "cpuUsageVal".

Mengapa kita memerlukan itu?

Jawabannya adalah karena fungsi "osu.cpuUsage" mengembalikan hasilnya melalui callback, sedangkan kita harus merespon request dengan JSON sesegera mungkin.

Oleh karena itulah, dalam kode tersebut, hasil dari "osu.cpuUsage" akan disimpan dalam variabel "cpuUsageVal" sementara JSON akan direspon sesegera mungkin

dengan:

```
res.json({name: "CPU Usage", value: cpuUsageVal});
```

Seandainya hasil dari "osu.cpuUsage" diberikan melalui return value, maka kita hanya perlu melakukan:

```
res.json({name: "CPU Usage", value: osu.cpuUsage ()});
```

Tapi tidak demikian bawaan dari package "os-utils" sehingga kita tidak melakukan itu.

Response untuk Mendapatkan File Snapshot

Respon yang terakhir ini agak rumit, tapi pada dasarnya kita hanya melakukan langkah-langkah ini:

1. Mendapatkan nilai-nilai system performance di saat user melakukan request
2. Mengubah nilai-nilai tersebut dalam bentuk JSON
3. Mengubah nilai JSON menjadi string
4. Menuliskan nilai tersebut dalam sebuah file bernama "snapshot.json"
5. Merespon request dengan downloadable content, yakni file "snapshot.json" tersebut

Perhatikanlah kode-kode ini:

```
app.get('/api/download_snapshot', function (req, res) {
  var snapshot = {
    when: new Date(),
    cpuUsage: cpuUsageVal,
    freememPercentage: osu.freememPercentage(),
    platform: osu.platform(),
    cpuCount: osu.cpuCount(),
    freeMem: osu.freemem(),
    totalMem: osu.totalmem(),
    sysUpTime: osu.sysUptime(),
    processUpTime: osu.processUptime()
  };

  var jstr = JSON.stringify(snapshot, null, '\n');
  fs.writeFile(__dirname + '/assets/snapshot.json', jstr, function(err) {
    if(err)
      return console.error(err);
    console.log('done');
```

```
    res.download(__dirname + '/assets/snapshot.json');
  });
});
```

Kita mendapatkan nilai-nilai system performance dan mengubahnya dalam format JSON di sini:

```
var snapshot = {
  when: new Date(),
  cpuUsage: cpuUsageVal,
  freememPercentage: osu.freememPercentage(),
  platform: osu.platform(),
  cpuCount: osu.cpuCount(),
  freeMem: osu.freemem(),
  totalMem: osu.totalmem(),
  sysUpTime: osu.sysUptime(),
  processUpTime: osu.processUptime()
};
```

Kemudian kita mengubah JSON tersebut menjadi string di sini:

```
var jstr = JSON.stringify(snapshot, null, '\n');
```

Argument pertama diisi oleh "var snapshot" yang merupakan JSON.

Argument ke-2 digunakan sebagai whitelist dari property JSON yang dimasukkan.

Dalam kasus ini, nilainya null, yang artinya semua property dimasukkan.

Argument ke-3, yakni dengan parameter '\n', adalah dengan apa whitespace dari string JSON tersebut diisi.

Dalam kasus ini, kita mengisinya dengan newline ('\n').

Kemudian kita menuliskan JSON string tersebut dalam sebuah file lalu memberikan downloadable content-nya kepada user:

```
fs.writeFile(__dirname + '/assets/snapshot.json', jstr, function(err) {
  if(err)
    return console.error(err);
  console.log('done');
```

```
res.download(__dirname + '/assets/snapshot.json');
});
```

Perhatikan pada:

```
fs.writeFile(__dirname + '/assets/snapshot.json'...
```

Kita menge-save snapshot tersebut pada folder yang telah diizinkan oleh server.

Kita melakukan itu karena user harus dapat mendownload file snapshot tersebut.

Mengimplementasikan Client Side Script pada File Index.html

Setelah semua request handler diimplementasikan, sekarang saatnya membuat client side script "index.html".

Karena kita telah membuat file-nya, berarti sekarang tinggal mengubahnya agar sesuai dengan kebutuhan aplikasi system monitor ini.

Membuat atau Mengcopy Favicon.png Ke Folder Assets

Karena kita akan menggunakan favicon di sisi client, maka Anda akan membutuhkan file PNG apapun dengan ukuran 16x16 pixel.

Letakkan file PNG tersebut dalam folder "assets/images".

Beri nama "favicon.png".

Mengimpor Front-End Javascript dan CSS

Hal pertama yang perlu dilakukan adalah mengimpor front-end javascript dan CSS.

Untuk mengimpor CSS:

```
<link rel="stylesheet" href="scripts/css/bootstrap.min.css">
<link rel="stylesheet" href="scripts/css/bootstrap-theme.min.css">
<link rel="stylesheet" href="assets/css/style.css">
```

Letakkan ini di antara tag "<head>" dan "</head>".

Sedangkan untuk mengimpor front-end javascript:

```
<script src="scripts/jquery.min.js"></script>
```



```
<script src="scripts/js/bootstrap.min.js"></script>
<script src="scripts/vue.js"></script>
<script src="scripts/vue-resource.min.js"></script>
<script src="scripts/Chart.js"></script>
```

Letakkan ini juga di antara tag "<head>" dan "</head>".

Perhatikan bahwa "vue.js", "vue-resource.min.js", dan "Chart.js" diimpor disini. Seperti yang telah saya isyaratkan di bagian awal tulisan ini.

Dengan demikian, keseluruhan file "index.html" adalah seperti ini:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1">

<title>Node.js Web System Monitor - Aplikasi System Monitor</title>

<meta name="description" content="Node.js Web System Monitor">
<meta name="author" content="Anonim">

<link rel="shortcut icon" type="image/png" href="assets/images/favicon.png"/>
<link rel="stylesheet" href="scripts/css/bootstrap.min.css">
<link rel="stylesheet" href="scripts/css/bootstrap-theme.min.css">
<link rel="stylesheet" href="assets/css/style.css">
<script src="scripts/jquery.min.js"></script>
<script src="scripts/js/bootstrap.min.js"></script>
<script src="scripts/vue.js"></script>
<script src="scripts/vue-resource.min.js"></script>
<script src="scripts/Chart.js"></script>
</head>
<body>
<div id="app" class="container">
<nav class="navbar navbar-default navbar-inverse">
<div class="container-fluid">
<div class="navbar-header">
<button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-
target="#navbar" aria-expanded="false" aria-controls="navbar">
<span class="sr-only">Toggle navigation</span>
<span class="icon-bar"></span>
```

```

<span class="icon-bar"></span>
<span class="icon-bar"></span>
</button>
<a class="navbar-brand" href="#">Node.js System Monitor</a>
</div>
<div id="navbar" class="navbar-collapse collapse">
<ul class="nav navbar-nav navbar-right">
<li><a href="#" v-on:click="home">Home</a></li>
<li><a href="#" v-on:click="about">About</a></li>
<li><a href="#" v-on:click="contact">Contact</a></li>
</ul>
</div>
</div>
</nav>

<div class="jumbotron">
<h1>{{ jumboMessage.title }}</h1>
<p>{{ jumboMessage.message }}</p>
</div>

<div class="row">
<div class="col-md-6">
<div class="panel panel-default">
<div class="panel-heading">
<b>Current {{ cpuUsage.name }}</b>
</div>
<div class="panel-body">
<canvas id="cpuUsageChartPie" width="100" height="50"></canvas>
</div>
</div>
</div>
<div class="col-md-6">
<div class="panel panel-default">
<div class="panel-heading">
<b>{{ cpuUsage.name }} Over Time</b>
</div>
<div class="panel-body">
<canvas id="cpuUsageChart" width="100" height="50"></canvas>
</div>
</div>
</div>
</div>

```

```

<div class="row">
<div class="col-md-6">
<div class="panel panel-default">
<div class="panel-heading">
<b>Current {{ memoryUsage.name }}</b>
</div>
<div class="panel-body">
<canvas id="memoryUsageChartPie" width="100" height="50"></canvas>
</div>
</div>
</div>

<div class="col-md-6">
<div class="panel panel-default">
<div class="panel-heading">
<b>{{ memoryUsage.name }} Over Time</b>
</div>
<div class="panel-body">
<canvas id="memoryUsageChart" width="100" height="50"></canvas>
</div>
</div>
</div>
</div>

<div class="row">
<div class="col-md-6">
<div class="panel panel-default">
<div class="panel-heading">
<b>{{ otherInformations.name }}</b>
</div>
<table class="table table-striped">
<tbody>
<tr>
<td><b>Platform</b></td>
<td>{{ otherInformations.value.platform }}</td>
</tr>
<tr>
<td><b>CPU Count</b></td>
<td>{{ otherInformations.value.cpuCount }}</td>
</tr>
<tr>
<td><b>Free Memory</b></td>

```

```

<td>{{ otherInformations.value.freeMem }} MB</td>
</tr>
<tr>
<td><b>Total Memory</b></td>
<td>{{ otherInformations.value.totalMem }} MB</td>
</tr>
<tr>
<td><b>System Uptime</b></td>
<td>{{ otherInformations.value.sysUpTime }} second(s)</td>
</tr>
<tr>
<td><b>Process Uptime</b></td>
<td>{{ otherInformations.value.processUpTime }} second(s)</td>
</tr>
</tbody>
</table>
</div>
</div>

<div class="col-md-6">
<div class="panel panel-default">
<div class="panel-heading">
<b>Tools</b>
</div>
<div class="panel-body text-center">
<a class="btn btn-primary btn-lg" href="/api/download_snapshot">Get Snapshot!</a>
</div>
</div>
</div>
</div>
</div>

<nav class="navbar navbar-default navbar-inverse">
<div class="container-fluid">
<p class="navbar-text pull-right">Copyright &#169; Anonim 2017</p>
</div>
</nav>
</div>
<script>
var app = new Vue({
el: '#app',
data: {
jumboMessage: {
title: "Home",

```

```

message: "Selamat Datang di Node.js Web System Monitor!"
},
cpuUsage: {
  name: "",
  value: ""
},
memoryUsage: {
  name: "",
  value: ""
},
otherInformations: {
  name: "",
  value: ""
}
},
created: function() {
  this.os_utils_cpu_usage();
  this.timer = setInterval(this.os_utils_cpu_usage, 1000);

  this.os_utils_memory_usage();
  this.timer = setInterval(this.os_utils_memory_usage, 1000);

  this.os_utils_others();
  this.timer = setInterval(this.os_utils_others, 1000);
},
methods: {
  home: function () {
    this.$http.get('/api/home').then(response => {
      this.jumboMessage = response.body;
    }, response => {

    });
  },
  about: function () {
    this.$http.get('/api/about').then(response => {
      this.jumboMessage = response.body;
    }, response => {

    });
  },
  contact: function () {
    this.$http.get('/api/contact').then(response => {

```

```

this.jumboMessage = response.body;
}, response => {

});
},
os_utils_cpu_usage: function () {
this.$http.get('/api/os_utils_cpu_usage').then(response => {
this.cpuUsage = response.body;
}, response => {

});
},
os_utils_memory_usage: function () {
this.$http.get('/api/os_utils_memory_usage').then(response => {
this.memoryUsage = response.body;
}, response => {

});
},
os_utils_others: function () {
this.$http.get('/api/os_utils_others').then(response => {
this.otherInformations = response.body;
}, response => {

});
}
});
</script>

<script>
var configCPUUsagePie = {
type: 'pie',
data: {
labels: [
"CPU Used",
"CPU Free"
],
datasets: [{
data: [100, 50],
backgroundColor: [
"#FF6384",
"#36A2EB"

```

```

],
hoverBackgroundColor: [
"#FF6384",
"#36A2EB"
]
}]
}
};

var cpuUsageChartPie = new
Chart(document.getElementById("cpuUsageChartPie").getContext("2d"),
configCPUUsagePie);

setInterval(function(){
configCPUUsagePie.data.datasets[0].data[0] = app.cpuUsage.value * 100;
configCPUUsagePie.data.datasets[0].data[1] = (1 - app.cpuUsage.value) * 100;
cpuUsageChartPie.update();
}, 1000);
</script>

<script>
var configCPUUsage = {
type: 'line',
data: {
labels: ["1", "2", "3", "4", "5", "6", "7", "8", "9", "10"],
datasets: [{
label: "CPU Usage Over Time",
data: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
fill: false
}]
}
};

var cpuUsageChart = new
Chart(document.getElementById("cpuUsageChart").getContext("2d"), configCPUUsage);

setInterval(function(){
configCPUUsage.data.datasets[0].data.shift();
configCPUUsage.data.datasets[0].data.push(app.cpuUsage.value * 100);
cpuUsageChart.update();
}, 1000);
</script>

```

```

<script>
var configMemoryUsagePie = {
type: 'pie',
data: {
labels: [
"Memory Used",
"Memory Free"
],
datasets: [{
data: [100, 50],
backgroundColor: [
"#FF6384",
"#36A2EB"
],
hoverBackgroundColor: [
"#FF6384",
"#36A2EB"
]
}]
}
};

var memoryUsageChartPie = new
Chart(document.getElementById("memoryUsageChartPie").getContext("2d"),
configMemoryUsagePie);

setInterval(function(){
configMemoryUsagePie.data.datasets[0].data[0] = app.memoryUsage.value * 100;
configMemoryUsagePie.data.datasets[0].data[1] = (1 - app.memoryUsage.value) * 100;
memoryUsageChartPie.update();
}, 1000);
</script>

<script>
var configMemoryUsage = {
type: 'line',
data: {
labels: ["1", "2", "3", "4", "5", "6", "7", "8", "9", "10"],
datasets: [{
label: "Memory Usage Over Time",
data: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
fill: false
}]
}
}

```



```

}
};

var memoryUsageChart = new
Chart(document.getElementById("memoryUsageChart").getContext("2d"),
configMemoryUsage);

setInterval(function(){
configMemoryUsage.data.datasets[0].data.shift();
configMemoryUsage.data.datasets[0].data.push(app.memoryUsage.value * 100);
memoryUsageChart.update();
}, 1000);
</script>
</body>
</html>

```

Penjelasan Kode dari File Index.html

Sekilas tidak ada yang aneh dengan file "index.html" karena hanya berupa file HTML biasa.

Tapi perhatikan baris-baris kode yang terdapat pada file "index.html" ini:

```

<b>Current {{ cpuUsage.name }}</b>

<b>{{ cpuUsage.name }} Over Time</b>

<b>Current {{ memoryUsage.name }}</b>

<b>{{ memoryUsage.name }} Over Time</b>

<b>{{ otherInformations.name }}</b>

<tr>
<td><b>Platform</b></td>
<td>{{ otherInformations.value.platform }}</td>
</tr>

<tr>
<td><b>CPU Count</b></td>
<td>{{ otherInformations.value.cpuCount }}</td>
</tr>

```

```

<tr>
<td><b>Free Memory</b></td>
<td>{{ otherInformations.value.freeMem }} MB</td>
</tr>

<tr>
<td><b>Total Memory</b></td>
<td>{{ otherInformations.value.totalMem }} MB</td>
</tr>

<tr>
<td><b>System Uptime</b></td>
<td>{{ otherInformations.value.sysUpTime }} second(s)</td>
</tr>

<tr>
<td><b>Process Uptime</b></td>
<td>{{ otherInformations.value.processUpTime }} second(s)</td>
</tr>

```

Di sana terdapat kode yang diapit oleh "{{" dan "}}". Ini bukan kode HTML bawaan.

Kode ini akan di-replace oleh Vue.js menjadi suatu nilai tertentu dari variable yang diletakkan di antara "{{" dan "}}".

Jadi, "{{ otherInformations.value.freeMem }}" misalnya, akan diisi nilai memory yang tidak terpakai yang didefinisikan di member variable "otherInformations.value.freeMem".

Lalu di mana "otherInformations" berada?

Jawabnya, variabel tersebut ada di dalam object Vue dari "vue.js" yang didefinisikan pada kode berikut ini:

```

var app = new Vue({
  el: '#app',
  data: {
    jumboMessage: {
      title: "Home",
      message: "Selamat Datang di Node.js Web System Monitor!"
    },
    cpuUsage: {

```

```

        name: "",
        value: ""
    },
    memoryUsage: {
        name: "",
        value: ""
    },
    otherInformations: { //<==DI SINI
        name: "",
        value: ""
    }
}
//....

```

Di samping itu, nilai "freeMem" dari "otherInformations.value.freeMem" didapatkan dari server melalui AJAX dalam fungsi ini:

```

os_utils_others: function () {
    this.$http.get('/api/os_utils_others').then(response => {
        this.otherInformations = response.body;
    }, response => {
    });
}

```

Perhatikan bahwa kita melakukan GET pada request handler "/api/os_utils_others" yang telah kita implementasikan sebelumnya.

Request tersebut direspon dengan JSON, kemudian nilainya masuk ke "otherInformations".

Akan tetapi itu semua hanya terjadi bila variable tersebut (yang diapit "{" dan "}") berada dalam tag HTML dengan ID "app".

Nilainya harus sama dengan nilai "el" dari object Vue:

```

var app = new Vue({
  el: '#app', //<==YANG INI

```

Maka Anda akan dapat membuktikan bahwa semua variable yang diapit dengan "{" dan "}" berada dalam tag HTML dengan ID "app" (merupakan child-nya), dalam hal ini berupa "div" tag:

```

<div id="app" class="container"> <!-- YANG INI -->

```

```
<nav class="navbar navbar-default navbar-inverse">
<div class="container-fluid">
```

Tag tersebut tepat berada dibawah tag "<body>" jika Anda ingin melihatnya.

Kita melakukan hal yang sama pada variable yang lain:

```
data: {
  jumboMessage: {
    title: "Home",
    message: "Selamat Datang di Node.js Web System Monitor!"
  },
  cpuUsage: {
    name: "",
    value: ""
  },
  memoryUsage: {
    name: "",
    value: ""
  },
  otherInformations: {
    name: "",
    value: ""
  }
},
```

Semua nilai ini dapat diterima dalam tag HTML dengan mengapitnya dengan "{{" dan "}}".

Hal ini disebut sebagai data binding.

Nilai awal dari data binding tersebut didapat ketika object Vue baru diciptakan, kemudian nilai update-nya diambil tiap satu detik dengan menggunakan "setInterval":

```
created: function() {
  this.os_utils_cpu_usage();
  this.timer = setInterval(this.os_utils_cpu_usage, 1000);
  this.os_utils_memory_usage();
  this.timer = setInterval(this.os_utils_memory_usage, 1000);
  this.os_utils_others(); //<==YANG INI
  this.timer = setInterval(this.os_utils_others, 1000); //<==KEMUDIAN YANG INI
```

```
},
```

Khusus untuk variable "cpuUsage" dan "memoryUsage", nilai data yang di-binding akan dimasukkan sebagai input bagi grafik pie dan line.

Yang pada akhirnya grafik tersebut juga akan di-update setiap satu detik dengan "setInterval".

Jadi ada 2 penggunaan "setInterval" di sini, yakni untuk request dan untuk update grafik.

Sebagai contoh, untuk grafik pie "cpuUsage":

```
<script>
var configCPUUsagePie = {
  type: 'pie',
  data: {
    labels: [
      "CPU Used",
      "CPU Free"
    ],
    datasets: [{
      data: [100, 50],
      backgroundColor: [
        "#FF6384",
        "#36A2EB"
      ],
      hoverBackgroundColor: [
        "#FF6384",
        "#36A2EB"
      ]
    }]
  }
};

var cpuUsageChartPie = new
Chart(document.getElementById("cpuUsageChartPie").getContext("2d"),
configCPUUsagePie);

setInterval(function(){ //<==SET INTERVAL LAGI
configCPUUsagePie.data.datasets[0].data[0] = app.cpuUsage.value * 100; //<==YANG INI
configCPUUsagePie.data.datasets[0].data[1] = (1 - app.cpuUsage.value) * 100;
//<==YANG INI
```

```
cpuUsageChartPie.update();
}, 1000);
</script>
```

Dan untuk grafik line cpuUsage:

```
<script>
var configCPUUsage = {
type: 'line',
data: {
labels: ["1", "2", "3", "4", "5", "6", "7", "8", "9", "10"],
datasets: [{
label: "CPU Usage Over Time",
data: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
fill: false
}]
}
};

var cpuUsageChart = new
Chart(document.getElementById("cpuUsageChart").getContext("2d"), configCPUUsage);

setInterval(function(){ //<==SET INTERVAL LAGI
configCPUUsage.data.datasets[0].data.shift();
configCPUUsage.data.datasets[0].data.push(app.cpuUsage.value * 100); //<==YANG
INI
cpuUsageChart.update();
}, 1000);
</script>
```

Hal yang serupa juga berlaku bagi "memoryUsage".

Bagian "Home", "About" dan "Contact" juga diperlakukan dengan cara yang serupa, hanya saja yang diambil nilainya hanya teks-nya saja dan nilainya tidak dimasukkan ke dalam grafik.

Agar user bisa berpindah dari "Home", ke "About", dan ke "Contact" maka digunakan event "v-on:click":

```
<ul class="nav navbar-nav navbar-right">
<li><a href="#" v-on:click="home">Home</a></li>
<li><a href="#" v-on:click="about">About</a></li>
```

```
<li><a href="#" v-on:click="contact">Contact</a></li>
```

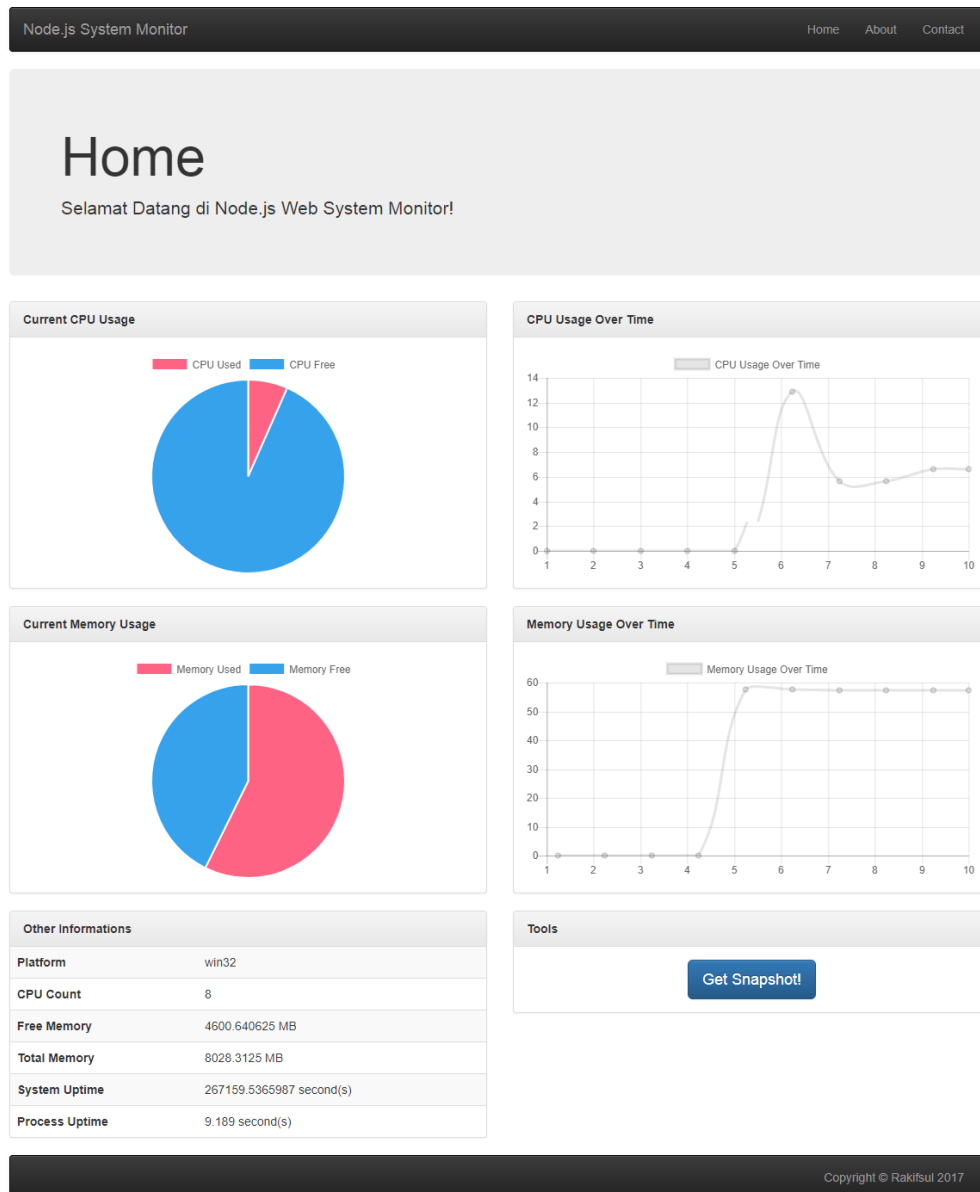
Jika link-link tersebut di-click, maka fungsi dari object Vue ini akan dipanggil:

```
methods: {  
  home: function () {  
    this.$http.get('/api/home').then(response => {  
      this.jumboMessage = response.body;  
    }, response => {  
  
    });  
  },  
  about: function () {  
    this.$http.get('/api/about').then(response => {  
      this.jumboMessage = response.body;  
    }, response => {  
  
    });  
  },  
  contact: function () {  
    this.$http.get('/api/contact').then(response => {  
      this.jumboMessage = response.body;  
    }, response => {  
  
    });  
  },  
}
```

Saat ini, kita telah selesai membuat aplikasi web system monitor. Hasil akhirnya dapat kita lihat di browser dengan mengakses:

```
http://localhost:5000/
```

Tampilannya kurang lebih seperti ini:



Tampilan Final

Apabila tombol Get Snapshot diklik, maka Anda akan mendapatkan file JSON berisi snapshot yang bisa di-download.

Penutup

Sekarang, kita telah mempelajari dasar-dasar Node.js yang walaupun singkat, tapi tetap menyeluruh.

Di samping itu, kita juga telah mencoba mempraktikkan dasar-dasar Node.js dalam pembuatan aplikasi sederhana di Bab 13 dan Bab 14.

Yang perlu Anda ketahui adalah, beberapa bab memiliki seluk-beluknya sendiri.

Satu bab dalam buku ini bisa dikembangkan menjadi banyak buku dan solusi jika kita mau mendalaminya.

Maka saya sangat merekomendasikan pembaca untuk mencoba mendalami bab yang paling diminati untuk dikembangkan menjadi sebuah buku atau solusi yang unik.

Dengan demikian, diharapkan ilmu tentang Node.js menjadi berkembang dan dapat menyelesaikan masalah yang belum ada solusinya.

Saya menyadari bahwa PASTI banyak kekurangan dalam tulisan saya ini.

Oleh karena itu, saya meminta maaf jika ada yang salah dalam buku ini.

Jika ada hal yang salah dalam buku ini, jangan diikuti, tapi perbaiki.

Daftar Pustaka

Bearnes, Brennen. 2016. How To Set Up a Node.js Application for Production on Ubuntu 16.04. <https://www.digitalocean.com/community/tutorials/how-to-set-up-a-node-js-application-for-production-on-ubuntu-16-04>. 18 Desember 2017. 18 Desember 2017.

Izalhidayat. 2015. Perbedaan Websocket dengan Ajax. <http://izalhidayat.student.telkomuniversity.ac.id/perbedaan-websocket-dengan-ajax/>. 18 Desember 2017.

Kantor, Ilya. t.t. The Modern JavaScript Tutorial. <https://javascript.info/>. 18 Desember 2017.

MapBox. 2017. node-sqlite3 README.md. <https://github.com/mapbox/node-sqlite3>. 18 Desember 2017.

MongoDB. 2017. node-mongodb-native Readme.md. <https://github.com/mongodb/node-mongodb-native>. 18 Desember 2017.

Mühler, Vincent. t.t. Tutorial to Native Node.js Modules with C++. Part 1 — An Introduction to Nan. <https://medium.com/netscape/tutorial-building-native-c-modules-for-node-js-using-nan-part-1-755b07389c7c>. 18 Desember 2017.

MySQLJS. 2017. mysql Readme.md. <https://github.com/mysqljs/mysql>. 18 Desember 2017.

Node.js Foundation. 2017. Express. <https://expressjs.com/>. 18 Desember 2017.

Node.js Foundation. t.t. Node.js v8.9.3 Documentation. <https://nodejs.org/dist/latest-v8.x/docs/api/>. 18 Desember 2017.

Pastebin. t.t. Pastebin.com - #1 paste tool since 2002!. <https://pastebin.com/>. 21 Desember 2017.

Pexel. t.t. Free stock photo of close-up, code, coding. <https://www.pexels.com/photo/close-up-code-coding-computer-239898/>. 23 Desember 2017.

Socket.io. t.t. Socket.IO. <https://socket.io/>. 18 Desember 2017.

Stack Exchange, Inc. 2017. Stack Overflow - Where Developers Learn, Share, & Build Careers. <https://stackoverflow.com/>. 22 Desember 2017.

Stefanov, Stoyan. 2006. 3 ways to define a JavaScript class.
<http://www.phpied.com/3-ways-to-define-a-javascript-class/>. 18 Desember 2017.

Traversy Media. 2017. Intro To JavaScript Unit Testing With Mocha JS & Chai.
<https://www.youtube.com/watch?v=MLTRHc5dk6s>. 18 Desember 2017.

TutorialsPoint. t.t. Learn Node.js. <https://www.tutorialspoint.com/nodejs/index.htm>.
18 Desember 2017.

TutorialsPoint. t.t. Learn Web Sockets. <http://www.tutorialspoint.com/websockets/>.
18 Desember 2017.

W3Schools. t.t. Node.js Tutorial. <https://www.w3schools.com/nodejs/> .

WebSocket. t.t. ws: a Node.js WebSocket library.
<https://github.com/websockets/ws>.

NAN, t.t. Native Abstractions for Node.js. <https://github.com/nodejs/nan>.

Lampiran

Buku ini dilengkapi dengan source code yang tersedia di folder Google Drive.

Source code untuk Bab 2 sampai Bab 12 memiliki nama folder yang sama dengan bab-nya.

Source code untuk Bab 13 memiliki nama folder "ls-node-js-pasta".

Source code untuk Bab 14 memiliki nama folder "ls-web-system-monitor".

Link download untuk source code buku ini adalah:

<http://bit.ly/2AMH7K1>