Aplikasi Web

Node.js

Equan Pr.

Table of Contents

Introduction 1.1

Node.js 1.2

- JavaScript Di Server 1.2.1
- Node.js In Action 1.2.2

Asinkron I/O & Event 1.3

- PHP & Server HTTP Apache 1.3.1
- Javascript & Node.js 1.3.2

Server HTTP Dasar 1.4

- Menjalankan Server 1.4.1

Server File Statis 1.5

Pemrosesan Data Form HTML 1.6

- URL Encode 1.6.1
- Multipart Data 1.6.2

Module npm 1.7

- Konsep 1.7.1
- Paket npm 1.7.2

ExpressJS 1.8

- Server File 1.8.1
  - Middleware 1.8.1.1
  - Akses Server 1.8.1.2
- Server REST 1.8.2

Database 1.9

- SQLite 1.9.1
  - Node Sqlite3 1.9.1.1
  - Enkripsi 1.9.1.2
    - sqlcipher 1.9.1.2.1
- MySQL 1.9.2
  - Node MySQL 1.9.2.1
- MongoDB 1.9.3
  - Node MongoDB 1.9.3.1

Mongoose 1.9.3.2
Testing 1.10
REST 1.10.1
Automasi 1.10.2
To Data URI 1.11
Penggunaan 1.11.1
todatauri.js 1.11.2
Koneksi MySQL 1.11.3
Person REST API 1.12
Cara Kerja 1.12.1
Server 1.12.2
Pengetesan 1.12.3
Image Uploader 1.13
Memakai ES6 1.14
Tentang Pengarang 1.15

Aplikasi Web Node.js

> **Note:** This book is written in Bahasa Indonesia and the main reason for that is because most of Node.js beginners in my country Indonesia is having difficulties to find Node.js resources written in my native language.

Buku ini cocok bagi siapa saja yang ingin mulai belajar pemrograman di platform Node.js khususnya untuk membangun aplikasi web. Syarat yang dibutuhkan adalah pembaca setidaknya pernah atau sudah bisa memakai bahasa pemrograman JavaScript.

Ebook ini bisa anda akses didua tempat yaitu:

- Github IDJS (versi online).
- Gitbook (versi online, PDF, Epub & Mobi).

Feedback

Untuk pertanyaan, kesalahan ketik atau permintaan silahkan mengisi Github issue.

Lisensi

Aplikasi Web Node.js oleh Equan Pr. di kerjakan di bawah lisensi Creative Commons Attribution-NonCommercial 4.0 International License.



Node.js

Javascript merupakan bahasa pemrograman yang lengkap hanya saja selama ini di pakai sebagai bahasa untuk pengembangan aplikasi web yang berjalan pada sisi client atau browser saja. Tetapi sejak ditemukannya Node.js oleh Ryan Dhal pada tahun 2009, Javascript bisa digunakan sebagai bahasa pemrograman di sisi server sekelas dengan PHP, ASP, C#, Ruby dll dengan kata lain Node.js menyediakan platform untuk membuat aplikasi Javascript dapat dijalankan di sisi server.

Untuk mengeksekusi Javascript sebagai bahasa server diperlukan engine yang cepat dan mempunyai performansi yang bagus. Engine Javascript dari Google bernama V8 yang dipakai oleh Node.js yang merupakan engine yang sama yang dipakai di browser Google Chrome.

JavaScript Di Server

Tak terelakkan bahwa Javascript merupakan bahasa pemrograman yang paling populer. Jika anda sebagai developer pernah mengembangkan aplikasi web maka penggunaan Javascript pasti tidak terhindarkan.

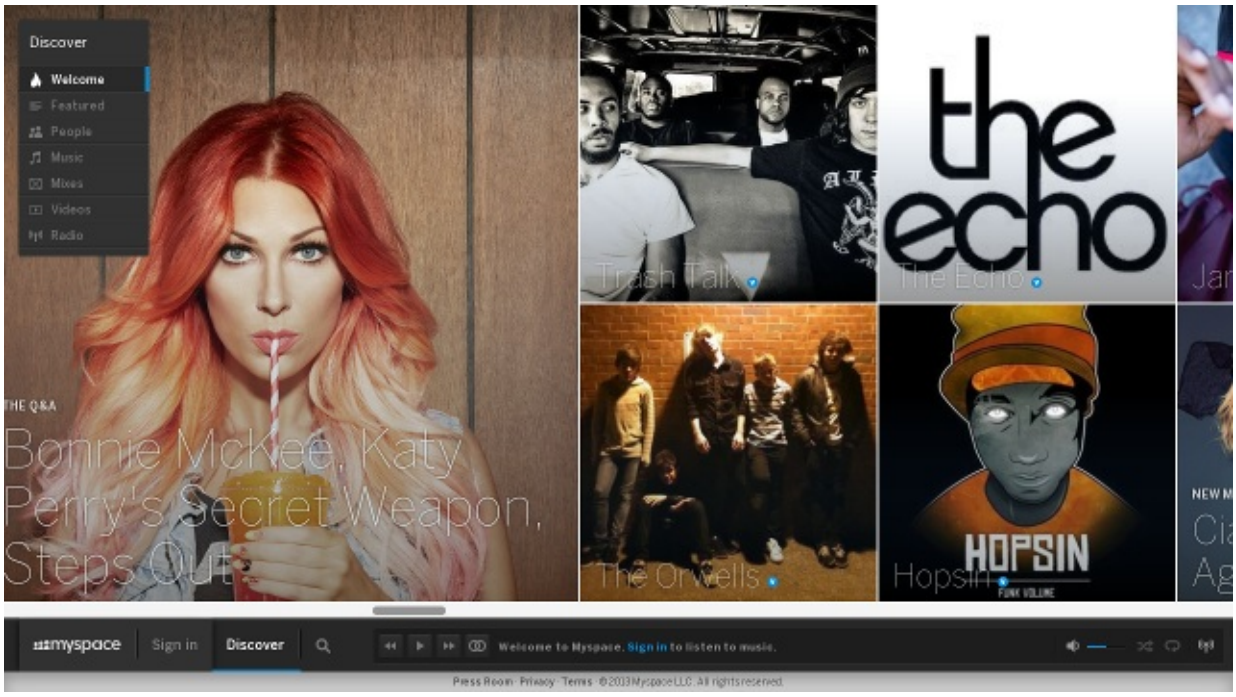
Sekarang dengan berjalannya Javascript di server lalu apa keuntungan yang anda peroleh dengan mempelajari Node.js, kurang lebih seperti ini :

- Pengembang hanya memakai satu bahasa untuk mengembangkan aplikasi lengkap client & server sehingga mengurangi *Learning Curve* untuk mempelajari bahasa server yang lain.
- Sharing kode antara client dan server atau istilahnya code reuse.
- Javascript secara native mendukung JSON yang merupakan standar transfer data yang banyak dipakai saat ini sehingga untuk mengkonsumsi data-data dari pihak ketiga pemrosesan di Node.js akan sangat mudah sekali.
- Database NoSQL seperti MongoDB dan CouchDB mendukung langsung Javascript sehingga interfacing dengan database ini akan jauh lebih mudah.
- Node.js memakai V8 yang selalu mengikuti perkembangan standar ECMAScript, jadi tidak perlu ada kekhawatiran bahwa browser tidak akan mendukung fitur-fitur di Node.js.

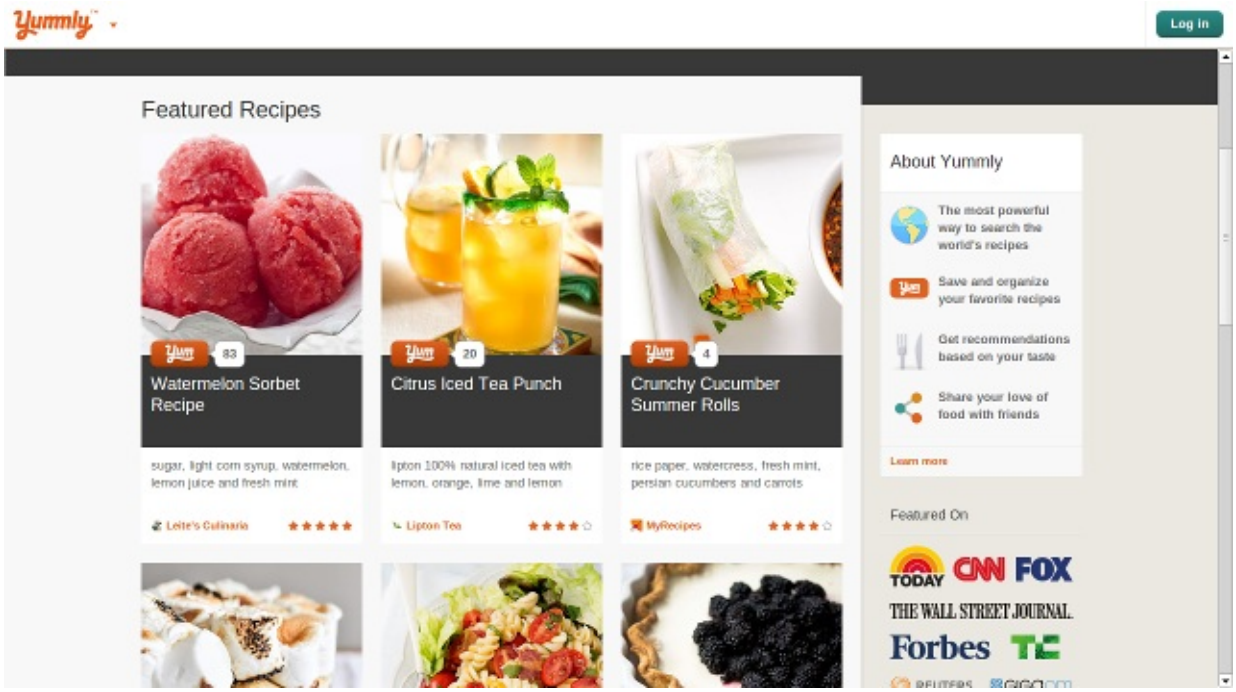
Node.js In Action

Supaya anda lebih tertarik dalam belajar Node.js berikut beberapa website terkenal yang sudah memakai Node.js

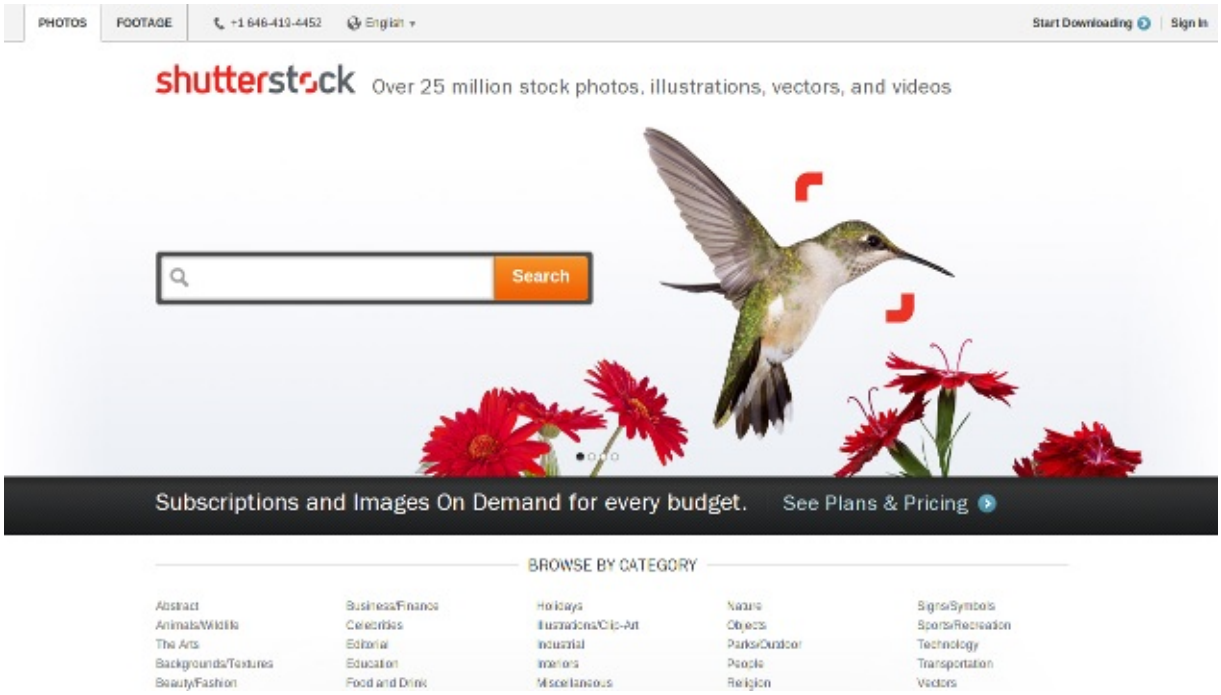
www.myspace.com



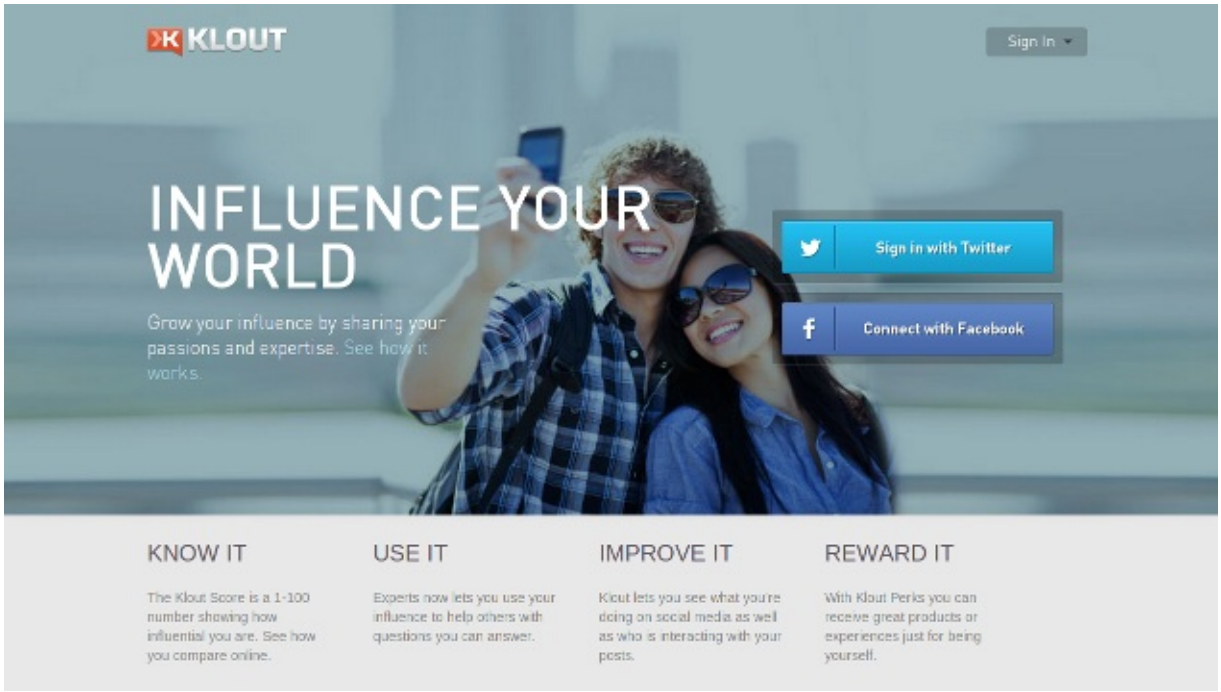
www.yummly.com



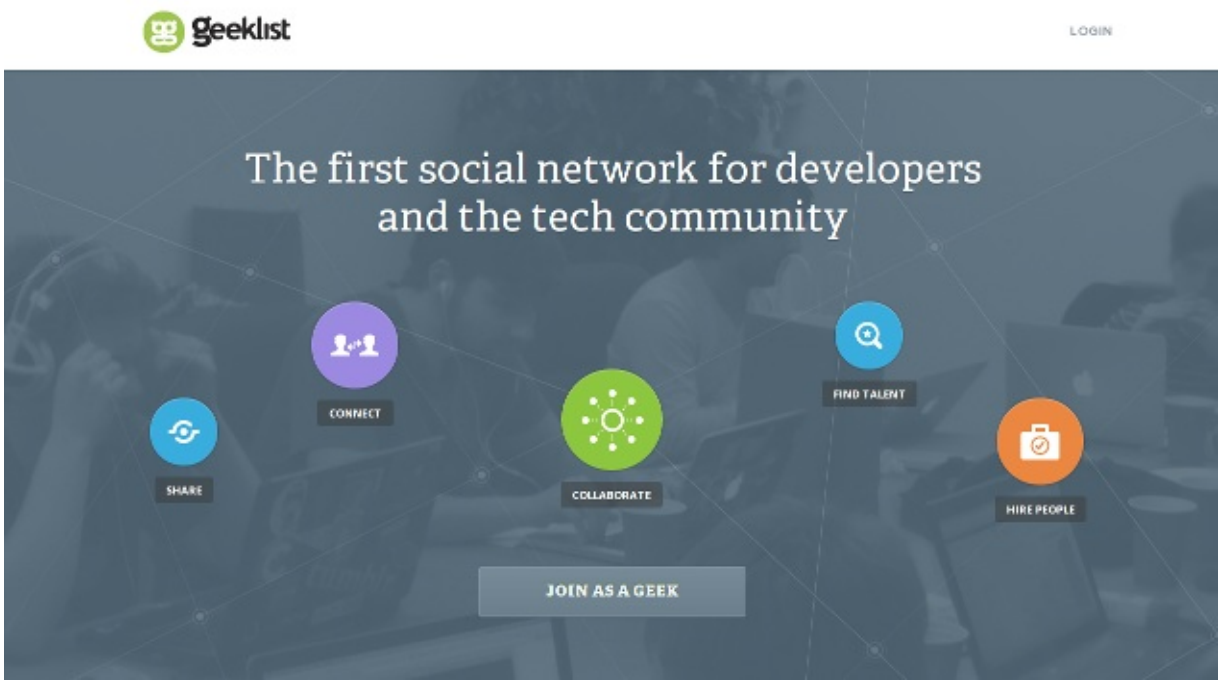
www.shutterstock.com



www.klout.com



www.geekli.st



www.learnboost.com



Apakah masih ragu untuk memakai Node.js ?...Kalau masih penasaran apa yang membuat Node.js berbeda dari backend pada umumnya, silahkan dilanjutkan membaca :smile:

Asinkron I/O & Event

Tidak seperti kebanyakan bahasa backend lainnya operasi fungsi di javascript lebih bersifat `asinkron` dan banyak menggunakan `event` demikian juga dengan Node.js. Sebelum penjelasan lebih lanjut mari kita lihat terlebih dahulu tentang metode `sinkron` seperti yang dipakai pada PHP dengan web server Apache.

PHP & Server HTTP Apache

Mari kita lihat contoh berikut yaitu operasi fungsi akses ke database MySQL oleh PHP yang dilakukan secara sinkron

```
$hasil = mysql_query("SELECT * FROM TabelAnggota");
print_r($hasil);
```

pengambilan data oleh `mysql_query()` diatas akan dijalankan dan operasi berikutnya `print_r()` akan diblok atau tidak akan berjalan sebelum akses ke database selesai. Yang perlu menjadi perhatian disini yaitu proses Input Output atau I/O akses ke database oleh `mysql_query()` dapat memakan waktu yang relatif mungkin beberapa detik atau menit tergantung dari waktu latensi dari I/O. Waktu latensi ini tergantung dari banyak hal seperti

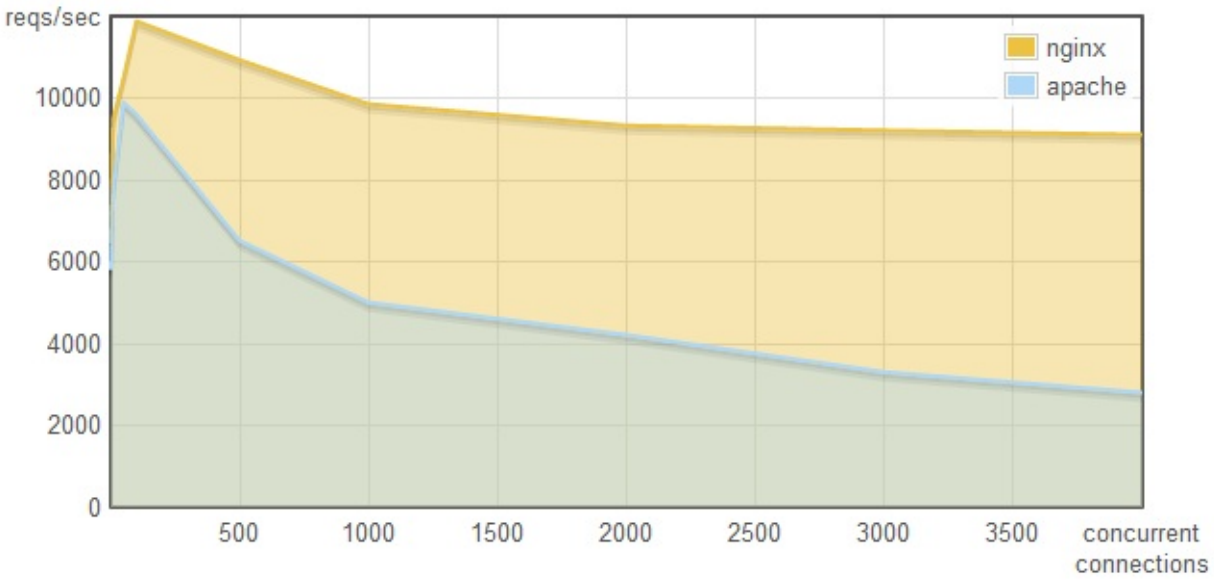
- Query database lambat akibat banyak pengguna yang mengakses
- Kualitas jaringan untuk akses ke database jelek
- Proses baca tulis ke disk komputer database yang membutuhkan waktu
- ...

Sebelum proses I/O selesai maka selama beberapa detik atau menit tersebut state dari proses `mysql_query()` bisa dibilang idle atau tidak melakukan apa-apa.

Lalu jika proses I/O di blok bagaimana jika ada request lagi dari user ? apa yang akan dilakukan oleh server untuk menangani request ini ?..penyelesaiannya yaitu dengan memakai pendekatan proses `multithread` . Melalui pendekatan ini tiap koneksi yang terjadi akan ditangani oleh `thread` . Thread disini bisa dikatakan sebagai task yang dijalankan oleh prosesor komputer.

Sepertinya permasalahan I/O yang terblok terselesaikan dengan pendekatan metode ini tetapi dengan bertambahnya koneksi yang terjadi maka `thread` akan semakin banyak sehingga prosesor akan semakin terbebani, belum lagi untuk switching antar thread menyebabkan konsumsi memory (RAM) komputer yang cukup besar.

Berikut contoh benchmark antara web server Apache dan Nginx (server HTTP seperti halnya Apache hanya saja Nginx memakai sistem asinkron I/O dan event yang mirip Node.js). Gambar ini diambil dari goo.gl/pvLL4



Bisa dilihat bahwa Nginx bisa menangani request yang jauh lebih banyak daripada web server Apache pada jumlah koneksi bersama yang semakin naik.

Javascript & Node.js

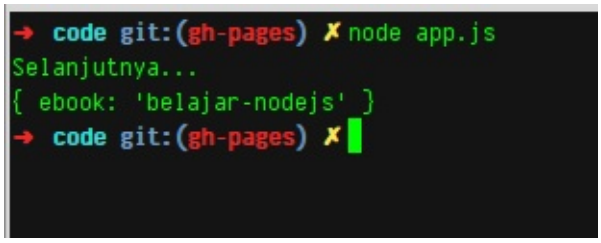
Kembali ke Javascript!. Untuk mengetahui apa yang dimaksud dengan pemrograman asinkron bisa lebih mudah dengan memakai pendekatan contoh kode. Perhatikan kode Javascript pada Node.js berikut

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

fs.readFile('./resource.json',function(err, data){
    if(err) throw err;
    console.log(JSON.parse(data));
});

console.log('Selanjutnya...');
```

fungsi `readFile()` akan membaca membaca isi dari file `resource.json` secara asinkron yang artinya proses eksekusi program tidak akan menunggu pembacaan file `resource.json` sampai selesai tetapi program akan tetap menjalankan kode Javascript selanjutnya yaitu `console.log('Selanjutnya...')`. Sekarng lihat apa yang terjadi jika kode javascript diatas dijalankan



Jika proses pembacaan file `resource.json` selesai maka fungsi callback pada `readFile()` akan di jalankan dan hasilnya akan ditampilkan pada console. Yah, fungsi callback merupakan konsep yang penting dalam proses I/O yang asinkron karena melalui fungsi callback ini data data yang dikembalikan oleh proses I/O akan di proses.

Lalu bagaimana platform Node.js mengetahui kalau suatu proses itu telah selesai atau tidak ?...jawabannya adalah Event Loop. Event - event yang terjadi karena proses asinkron seperti pada fungsi `fs.readFile()` akan ditangani oleh yang namanya Event Loop ini.

Campuran teknologi antara event driven dan proses asinkron ini memungkinkan pembuatan aplikasi dengan penggunaan data secara masif dan real-time. Sifat komunikasi Node.js I/O yang ringan dan bisa menangani user secara bersamaan dalam jumlah relatif besar tetapi tetap menjaga state dari koneksi supaya tetap terbuka dan dengan penggunaan memori yang cukup kecil memungkinkan pengembangan aplikasi dengan penggunaan data yang besar dan kolaboratif...Yeah, Node.js FTW! :metal:

Server HTTP Dasar

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

`server-http.js`

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

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

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

server.listen(PORT);

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

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

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

Tiap request yang terjadi akan ditangani oleh fungsi callback `requestListener` . Cara kerja callback ini hampir sama dengan ketika kita menekan tombol button html yang mempunyai atribut event `onclick` , jika ditekan maka fungsi yang teregistrasi oleh event `onclick` yaitu `clickHandler(event)` akan dijalankan.

`onclick-button.html`

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

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

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

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

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

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

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

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

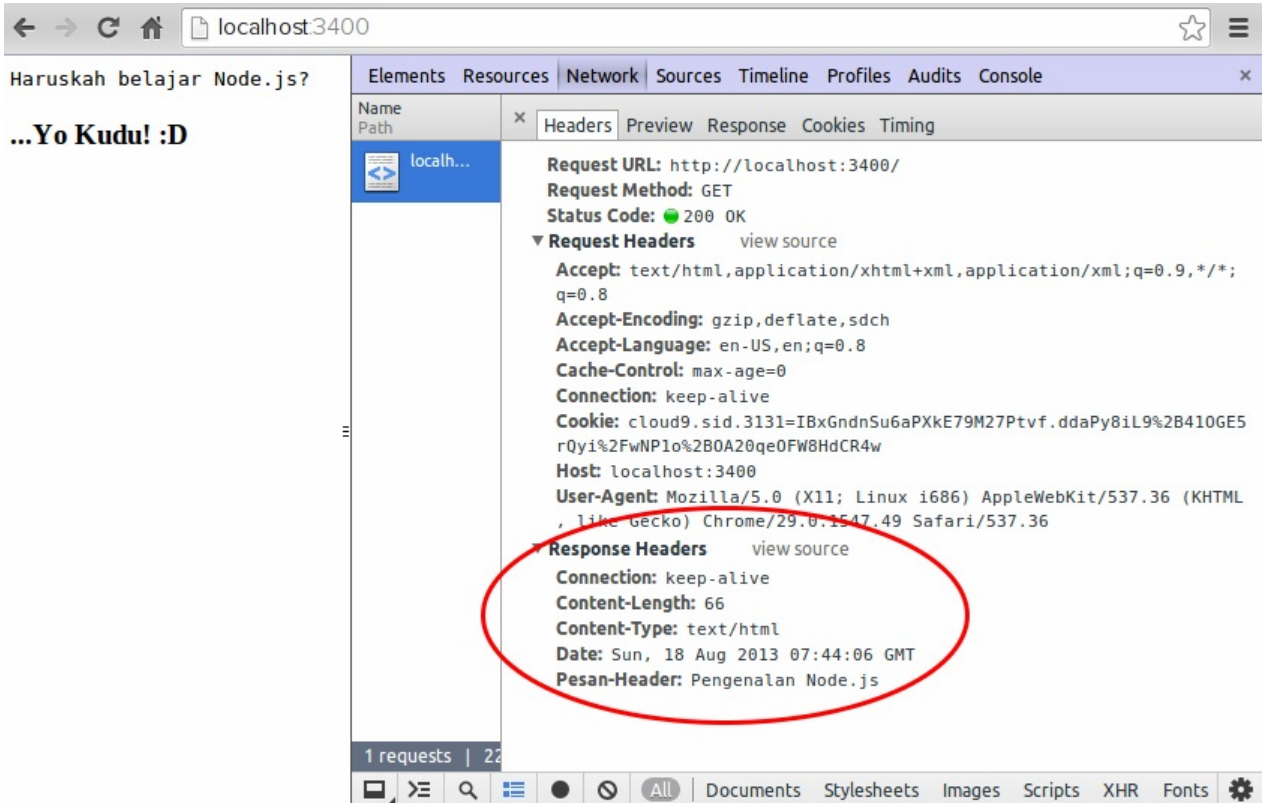
Satu hal lagi yaitu tentang kode status dari respon HTTP. Kode status ini bisa disetting selain 200 (request http sukses), misalnya bila diperlukan halaman error dengan kode status 404.

Menjalankan Server

Untuk menjalankan server Node.js ketik perintah berikut di terminal

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

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



Server File Statis

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

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

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

`server-file.js`

```
var http = require('http'),
    parse = require('url').parse,
    join = require('path').join,
    fs = require('fs'),
    root = join(__dirname, 'www'),
    PORT = 3300,

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

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

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

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

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

    });

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

Berikut sedikit penjelasan dari kode diatas

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

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

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

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

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

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

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

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

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

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

- Node.js API Stream
- Stream Handbook

Pemrosesan Data Form HTML

Aplikasi web memperoleh data dari pengguna umumnya melalui pengisian form HTML. Contohnya seperti ketika registrasi di media sosial atau aktifitas update status di Facebook misalnya pasti akan mengisi text field dan kemudian menekan tombol submit ataupun enter agar data bisa terkirim dan diproses oleh server.

Data yang dikirim oleh form biasanya salah satu dari dua tipe mime berikut

- application/x-www-form-urlencoded
- multipart/form-data

Node.js sendiri hanya menyediakan parsing data melalui `body` dari `request` sedangkan untuk validasi atau pemrosesan data akan diserahkan kepada komunitas.

URL Encode

Hanya akan dibahas untuk 2 metode HTTP yaitu `GET` dan `POST` saja. Metode `GET` untuk menampilkan form html dan `POST` untuk menangani data form yang dikirim

Ok langsung kita lihat kode server sederhana untuk parsing data form melalui objek `request` dengan data form bertipe `application/x-www-form-urlencoded` yang merupakan default dari tag form.

```
var http = require('http');
var data = [];
var qs = require('querystring');

var server = http.createServer(function(req, res){
    if('/' == req.url){
        switch(req.method){
            case 'GET':
                tampilkanForm(res);
                break;
            case 'POST':
                prosesData(req, res);
                break;
            default:
                badRequest(res);
        }
    } else {
        notFound(res);
    }
});

function tampilkanForm(res){
    var html = '<html><head><title>Data Hobiku</title></head><body>'
        + '<h1>Hobiku</h1>'
        + '<form method="post" action="/">'
        + '<p><input type="text" name="hobi"></p>'
        + '<p><input type="submit" value="Simpan"></p>'
        + '</form></body></html>';

    res.setHeader('Content-Type', 'text/html');
    res.setHeader('Content-Length', Buffer.byteLength(html));
    res.end(html);
}

function prosesData(req, res) {
   var body= '';
   req.setEncoding('utf-8');
   req.on('data', function(chunk){
       body += chunk;
```

```
    });

    req.on('end', function(){
        var data = qs.parse(body);
        res.setHeader('Content-Type', 'text/plain');
        res.end('Hobiku: '+data.hobi);
    });
}

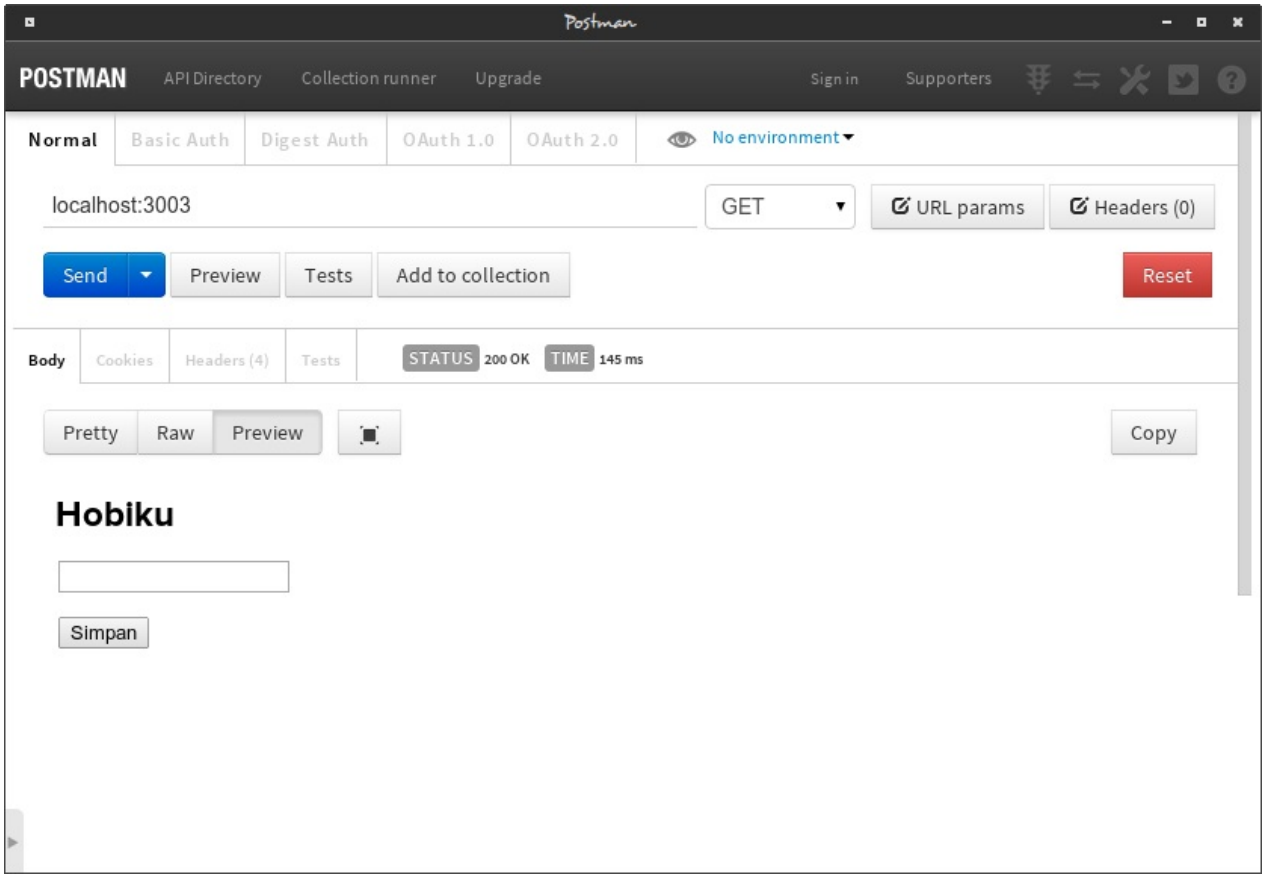
function badRequest(res){
    res.statusCode = 400;
    res.setHeader('Content-Type', 'text/plain');
    res.end('400 - Bad Request');
}

function notFound(res) {
    res.statusCode = 404;
    res.setHeader('Content-Type', 'text/plain');
    res.end('404 - Not Found');
}

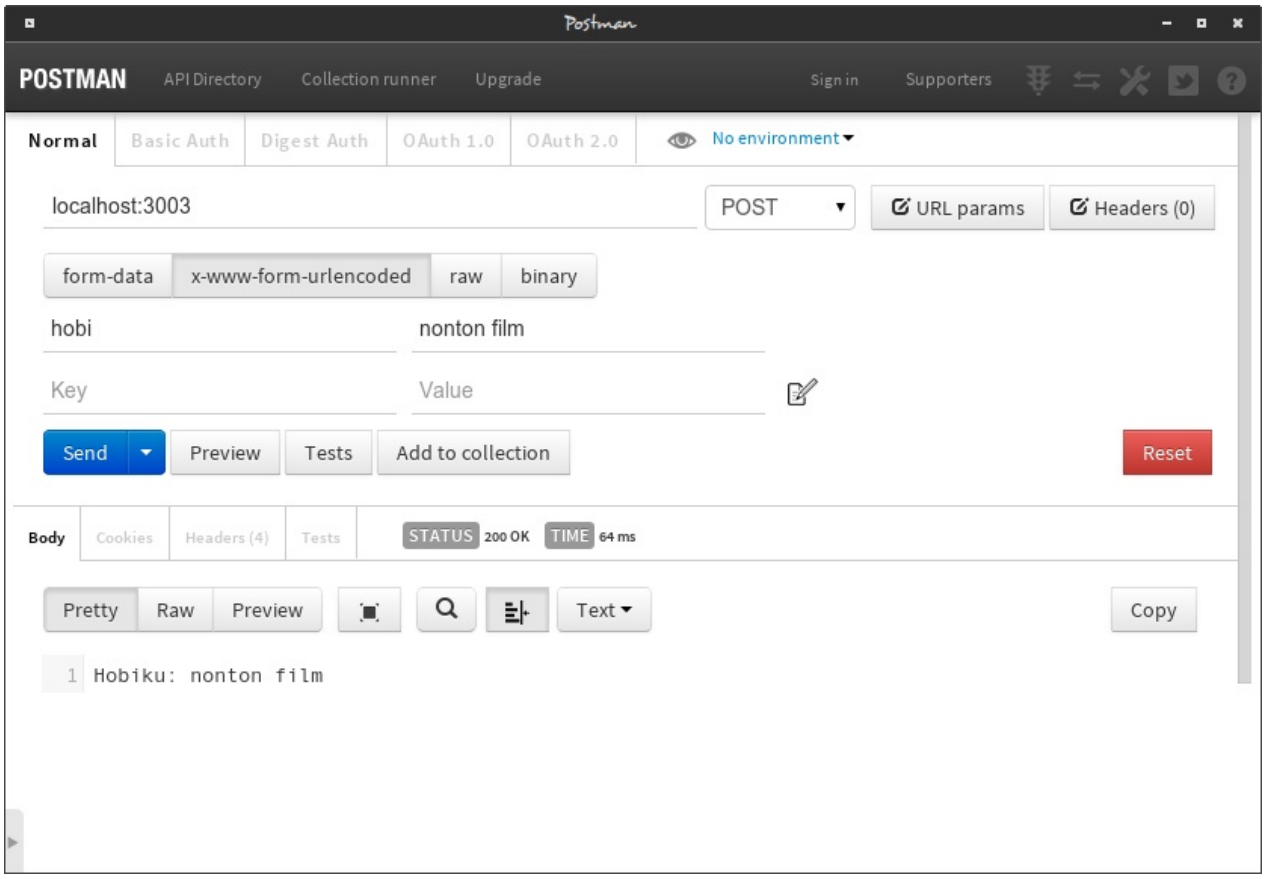
server.listen(3003);
console.log('server http berjalan pada port 3003');
```

Untuk mengetest `GET` dan `POST` bisa dilakukan melalui curl, browser atau melalui gui Postman.

GET


Postman
POSTMAN
API Directory
Collection runner
Upgrade
Sign in
Supporters
Normal
Basic Auth
Digest Auth
OAuth 1.0
OAuth 2.0
No environment
localhost:3003
GET
URL params
Headers (0)
Send
Preview
Tests
Add to collection
Reset
Body
Cookies
Headers (4)
Tests
STATUS 200 OK
TIME 145 ms
Pretty
Raw
Preview
Copy
Hobiku
Simpan

POST


Postman
POSTMAN
API Directory
Collection runner
Upgrade
Sign in
Supporters
Normal
Basic Auth
Digest Auth
OAuth 1.0
OAuth 2.0
No environment
localhost:3003
POST
URL params
Headers (0)
form-data
x-www-form-urlencoded
raw
binary
hobi
nonton film
Key
Value
Send
Preview
Tests
Add to collection
Reset
Body
Cookies
Headers (4)
Tests
STATUS 200 OK
TIME 64 ms
Pretty
Raw
Preview
Text
Copy
1 Hobiku: nonton film

module `querystring` dari Node.js berfungsi untuk mengubah url encoded string menjadi objek JavaScript. Contohnya

```
querystring.parse('nama=lanadelrey&job=singer&album=borntodie&ultraviolence');
// returns
{ nama: 'lanadelrey', job: 'singer',album: ['borntodie', 'ultraviolence']}
```

Multipart Data

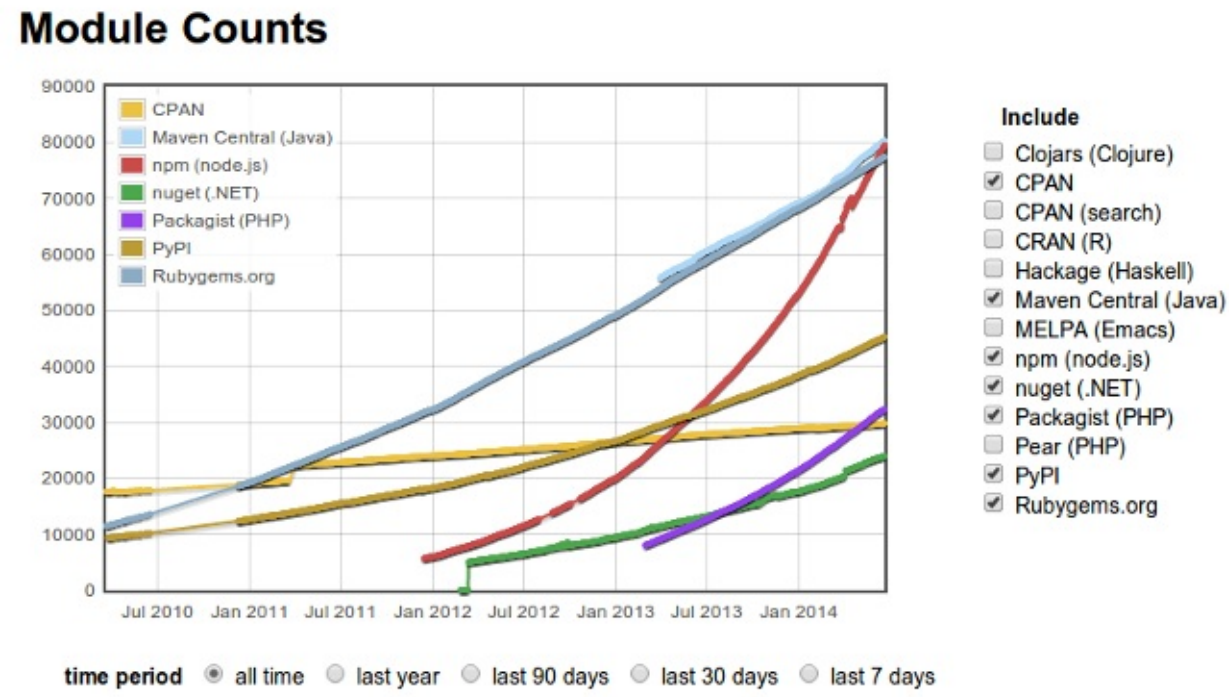
Parsing data form dan file yang di *upload* ke server Node.js adalah pekerjaan yang cukup susah kalo dikerjakan secara manual atau dari *scratch* karena disamping harus *parsing* data binary yang berupa file juga harus *parsing* data form.

Protokol untuk `multipart/form-data` di definisikan secara lengkap di RFC 2388.

Module Npm

```
nice people matter
```

npm merupakan package manager untuk Node.js dan untuk nama `npm` bukanlah suatu singkatan. Hanya dalam waktu 2 tahun sejak di releasenya Node.js ke publik jumlah modul melesat jauh bahkan hampir menyamai modul java ataupun ruby gems.



Grafik diatas didapat dari http://modulecounts.com.

Banyaknya push module ke repositori npm dapat diartikan adanya kepercayaan publik terhadap platform ini dan untuk kedepannya Node.js akan menjadi platform yang prospektif untuk berinvestasi.

Konsep

npm memakai sistem modul CommonJS yang cukup mudah dalam penggunaanya. Sistem modul ini akan meng-export objek JavaScript ke variabel `exports` yang bersifat global di modul tersebut.

Sebagai contoh

`band.js`

```
'use strict';

function Band(){}

Band.prototype.info = function(){
    return 'Nama Band: '+this.name;
}

Band.prototype.add = function(name){
    this.name = name;
}

module.exports = new Band();
```

Untuk pemakaiannya seperti di bawah ini

`app.js`

```
var band = require('./band.js');
band.add('Dewa 19');

console.log(band.info);
```

`require()` diatas adalah fungsi sinkron yang meload paket atau modul lain dari sistem file.

Paket npm

Secara default data paket npm disimpan di registry npmjs.org. Sehingga untuk menginstall paket npm tertentu anda bisa mencari paket ini melalui command `npm` atau langsung melalui website.

Sejak versi Node.js 0.6.3 command `npm` sudah ter-bundle dengan installer Node.js. Untuk menginstall modul npm yang anda butuhkan ketik misalnya

```
npm install express
```

perintah diatas akan mendownload paket `express` dari `http://npmjs.org` dan secara otomatis akan membuat directory `node_modules` .

Untuk memakai modul `express` ini cukup dengan membuat file JavaScript baru di luar direktori `node_modules` dan load modul dengan keyword `require` .

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

// kode lainnya
```

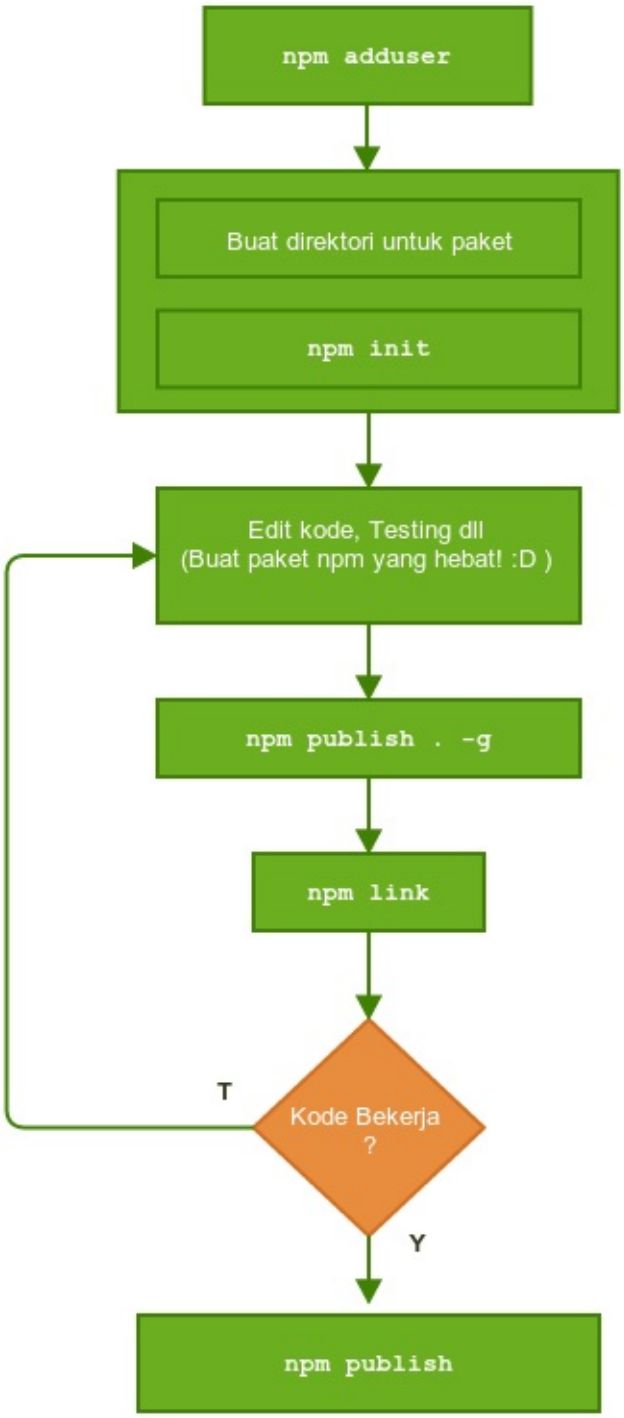
Membuat Paket npm

Sebelum membuat paket npm pastikan fungsionalitas yang anda cari tidak ada dalam registry npm. Caranya yaitu anda bisa menggunakan perintah

```
npm search
```

atau dengan memakai website berikut npmjs.org, node-modules.com atau npmsearch.com

Untuk membuat paket npm caranya cukup mudah. Berikut alur umum untuk membuat paket npm untuk di publish ke registry `npmjs.org` .



Secara garis besar proses pembuatan paket npm menurut alur diatas akan dijelaskan sebagai berikut

Registrasi

Sebelum publish ke registry npmjs.org kita harus registrasi dulu melalui perintah berikut

```
npm adduser
```

Buat Project

Untuk membuat project baru dari nol langkah pertama adalah membuat direktori

```
mkdir npmproject
```

Kemudian inisialisasi project tersebut

```
npm init
```

perintah diatas akan membuat file `package.json` yang isinya adalah info dan dependensi project. Ikuti saja tiap pertanyaan dan isi informasi sesuai dengan paket yang ingin anda buat.

Contohnya pada paket `svh` berikut ini

`package.json`

```
{
  "name": "svh",
  "version": "0.0.7-beta",
  "author": "Equan Pr.",
  "description": "Simple file server for html-javascript web client app development",
  "keywords": [
    "process",
    "reload",
    "watch",
    "development",
    "restart",
    "server",
    "monitor",
    "auto",
    "static",
    "nodemon"
  ],
  "homepage": "https://github.com/junwatu/svh",
  "bugs": "https://github.com/junwatu/svh/issues",
  "main": "./lib/core.js",
  "scripts": {
    "test": "./node_modules/mocha/bin/mocha"
  },
  "dependencies": {
    "async": "~0.2.9",
    "chalk": "0.2.x",
    "cheerio": "0.12.x",
```

```
    "commander": "2.0.x",
    "compression": "^1.0.2",
    "concat-stream": "1.0.x",
    "express": "4.x",
    "morgan": "^1.1.1",
    "send": "^0.3.0",
    "watch": "0.8.x",
    "wordgenerator": "0.0.1"
  },
  "devDependencies": {
    "gulp": "^3.5.6",
    "gulp-uglifyjs": "^0.3.0",
    "gulp-util": "2.2.14",
    "mocha": "1.13.x",
    "supertest": "0.8.x"
  },
  "repository": {
    "type": "git",
    "url": "http://github.com/junwatu/svh.git"
  },
  "bin": {
    "svh": "./bin/svh"
  },
  "license": "MIT"
}
```

Publish Lokal

Sebelum di publish pastikan paket anda bisa berjalan atau digunakan pada komputer lokal. Perintah berikut akan menginstall paket anda secara global di komputer.

```
npm publish . -g
```

atau jika diinginkan link simbolik bisa memakai perintah npm berikut

```
npm link
```

Publish Publik

```
npm publish
```

Untuk lebih jelasnya silahkan kunjungi dokumentasi untuk developer npm.

ExpressJS

Untuk memudahkan pembuatan aplikasi web anda bisa menggunakan framework ExpressJS daripada harus menggunakan module http bawaan Node.js. Framework ini menawarkan beberapa fitur seperti routing, rendering view dan mendukung middleware dengan kata lain anda akan banyak menghemat waktu dalam pengembangan aplikasi Node.js.

ExpressJS merupakan framework minimal yang sangat fleksibel. Anda bisa membuat web server HTML, server file statik, aplikasi chat, search engine, sosial media, layanan web dengan akses melalui REST API atau aplikasi hybrid yaitu selain pengguna mempunyai akses melalui REST API juga mempunyai akses ke HTML page.

Server File

Sebelumnya bikin project kecil dengan mengetikkan perintah berikut pada direktori project

```
$ npm init
```

dan berikan nama project sebagai `server-file-statik` . Direktori project dari server file bisa dilihat pada sususan tree dibawah ini.

> Susunan file dan direktori

```
    server-file-statis
    ├── package.json
    ├── app.js
    ├── node_modules
    └── publik
        ├── index.html
```

Dengan adanya npm instalasi ExpressJS sangat mudah, ketik perintah berikut pada direktori project.

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

> Catatan :
>
> Perintah diatas akan menginstall ExpressJS dengan versi yang paling terbaru (pada saat buku ini ditulis versi terbaru adalah 4.x). Jika membutuhkan versi tertentu cukup dengan menambahkan '@' dan nomer versi yang akan di inginkan seperti contoh berikut
>
> ```
>    $ npm install express@3  --save
> ```

Untuk kedepannya bahasan mengenai ExpressJS ini akan memakai versi 4.x.

Kode

Jika anda ingat server file yang memakai modul http pada bab sebelumnya berikut merupakan versi yang memakai ExpressJS

> app.js

```
'use strict';

var express = require('express');
var server = express();
var logger = require('morgan');

server.use(logger('dev'));

server.use(express.static(__dirname+'/publik'));

server.listen(4000, function(){
    console.log('Server file sudah berjalan bos!');
});
```

Seperti yang dijelaskan pada bab sebelumnya untuk memakai module Node.js di gunakan keyword `require`.

Modul `express` akan menangani tiap request dari user dan kemudian akan memberikan response berupa file yang diinginkan. Pada kode diatas file yang akan diberikan ke pengguna disimpan pada folder `publik`.

Middleware

Fungsionalitas yang diberikan oleh ExpressJS di bantu oleh yang namanya `middleware` yaitu fungsi asinkron yang bisa mengubah request dan respon di server.

Pada kode diatas contoh middleware yaitu modul `morgan` . Cara pemakaian `middleware` yaitu melalui `app.use()` .

Module `morgan` merupakan modul untuk logger yang berfungsi untuk pencatatan tiap request ke server. Pencatatan ini atau istilahnya `logging` akan ditunjukkan di console terminal.

Untuk menginstall modul ini ketik perintah berikut

```
$ npm install morgan --save
```

Middleware middleware ini bisa anda lihat di link berikut

https://github.com/senchalabs/connect#middleware

Akses Server

Untuk menjalankan server file statik ini

```
$ node app.js
```

Anda bisa meletakkan file apa saja pada direktori 'publik'. Untuk mengakses server ini ketik alamat berikut pada browser

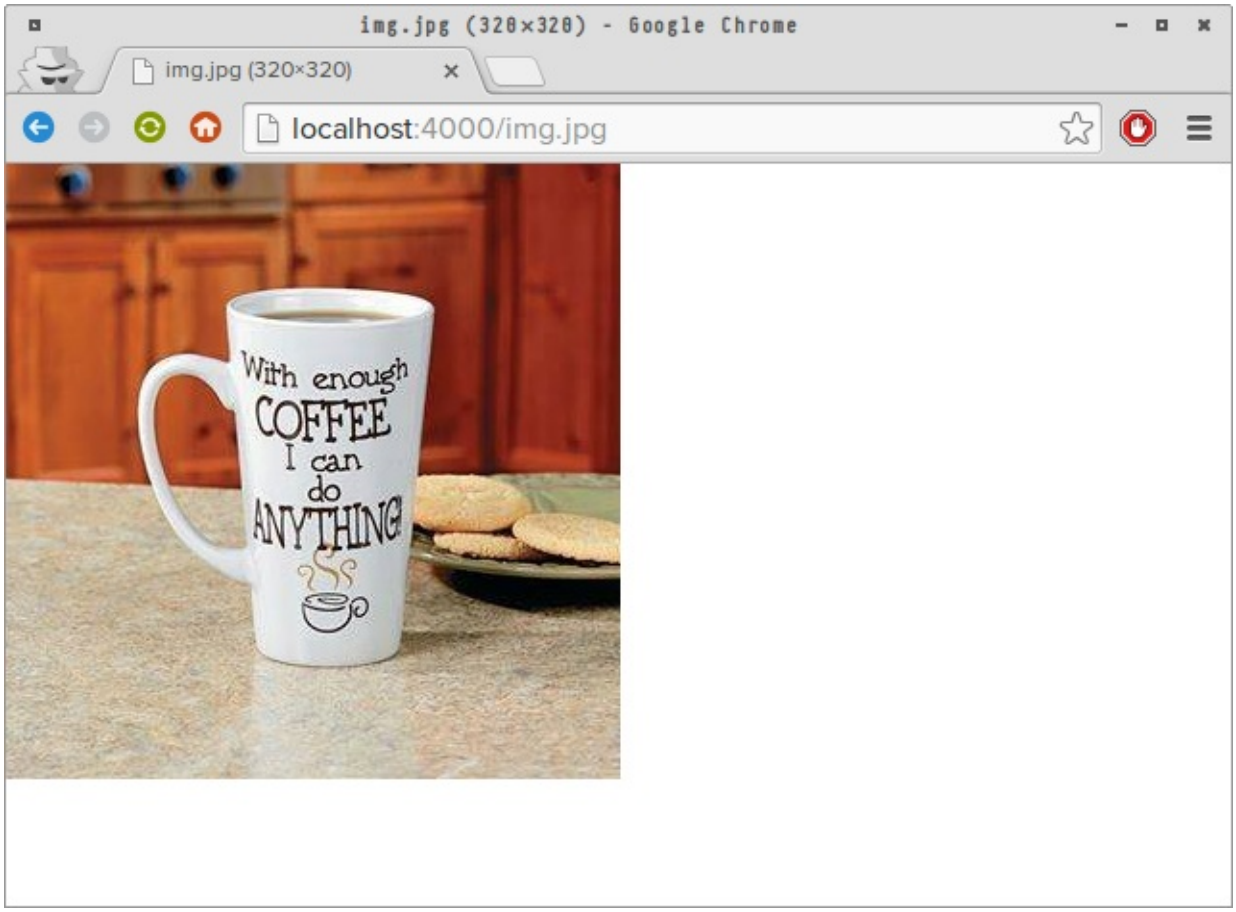
```
http://localhost:4000
```

dan secara default akan menampilkan file `index.html` di browser.

Untuk mengakses file yang lain format URL adalah

```
http://localhost:4000/[nama-file]
```

misalnya untuk mengakses file `img.jpg`



Server REST

Framework ExpressJS sangat banyak digunakan sebagai aplikasi RESTful. Bagi anda yang belum tahu, REST atau Representational State Transfer adalah arsitektur yang digunakan dalam desain aplikasi network. Idenya yaitu daripada memakai teknik seperti CORBA, SOAP atau RPC untuk membuat Web Service lebih baik jika memakai protokol HTTP yang lebih mudah.

Aplikasi RESTful memakai request HTTP untuk operasi Create, Read, Update dan Delete data. REST itu sendiri bukanlah suatu standard jadi tidak ada yang namanya spec dari W3C. Sehingga sangat mudah bagi bahasa pemrograman untuk menggunakan arsitektur ini. Keuntungan lainnya yaitu dengan arsitektur ini bisa dibangun berbagai macam teknologi klien seperti web atau mobile ataupun dekstop untuk mengakses aplikasi RESTful tersebut.

Untuk contoh aplikasi RESTful memakai ExpressJS akan diberikan pada Bab **Person REST API**.

Database

Berkembang pesatnya komunitas Node.js menyebabkan dukungan pustaka yang semakin cepat juga. Dengan waktu yang relatif cepat platform ini sekarang mendukung banyak tipe database seperti SQLite, MySQL, MongoDB, Redis, CouchDB dll. Dalam buku ini hanya akan dibahasa 3 database saja yaitu SQLite, MySQL dan MongoDB.

SQLite

Mungkin database ini sudah tidak asing lagi bagi developer seperti anda karena memang banyak digunakan dalam aplikasi portable dan embeddable.

Kekuatan SQLite secara garis besar sebagai berikut

Serverless

SQLite tidak memerlukan proses lain untuk beroperasi seperti pada database lain yang umumnya sebagai server. Library SQLite akan mengakses file data secara langsung.

Zero Configuration

Karena sifatnya bukan server maka database ini tidak memerlukan setup tertentu. Untuk membuat database cukup seperti ketika membuat file baru.

Cross-Platfrom

Semua data tersimpan pada satu sistem file yang bersifat cross platform dan tidak memerlukan administrasi.

Self-Contained

Library SQLite sudah mencakup semua sistem database sehingga dengan mudah dapat diintegrasikan ke aplikasi host.

Small Runtime Footprint

Ukuran default dari SQLite ini kurang dari 1MB dan membutuhkan hanya beberapa Megabyte memory.

Transactional

Operasi transaksi kompatibel dengan ACID sehingga aman kalau harus mengakses data dari proses banyak thread.

Untuk penjelasan lebih detil, silahkan kunjungi website resmi http://sqlite.org

node sqlite3

Komunitas node menyediakan banyak solusi untuk memakai SQLite di platform Node.js. Salah satu yang paling populer adalah modul `node-sqlite3`.

Instal

node `sqlite3` merupakan binding modul JavaScript yang asinkron untuk database sqlite3.

Untuk penginstalan modul ini ketik perintah berikut

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

CRUD

Operasi database seperti **Create**, **Read**, **Update** dan **Delete** untuk database SQLite sangat mudah seperti pada contoh berikut

`app.js`

```
/**
 * Akses SQLite 3
 */
var sqlite = require('sqlite3').verbose();
var file = 'film.db';
var db = new sqlite.Database(file);
var fs = require('fs');

// Sample Data
var film = {
    judul: "Keramat",
    release: "2009",
    imdb: "http://www.imdb.com/title/tt1495818/",
    deskripsi: "Film horror paling horror!"
}

var filmUpdate = {
    id: 1,
    deskripsi: "Best Indonesian Horror Movie."
}

// SQL Statement
var CREATE_TABLE = "CREATE TABLE IF NOT EXISTS fdb ( id INTEGER PRIMARY KEY AUTOINCREM
```

```
ENT, judul TEXT NOT NULL, release TEXT NOT NULL, imdb TEXT, deskripsi TEXT )";
var INSERT_DATA = "INSERT INTO fdb (judul, release, imdb, deskripsi) VALUES (?, ?, ?,
?)";
var SELECT_DATA = "SELECT * FROM fdb";
var UPDATE_DATA = "UPDATE fdb SET deskripsi=$deskripsi WHERE id=$id";

// Run SQL one at a time
db.serialize(function() {
    // Create table
    db.run(CREATE_TABLE, function(err) {
        if (err) {
            console.log(err);
        } else {
            console.log('CREATE TABLE');
        }
    });

    // Insert data with sample data
    db.run(INSERT_DATA, [film.judul, film.release, film.imdb, film.deskripsi], functio
n(err) {
        if (err) {
            console.log(err);
        } else {
            console.log('INSERT DATA');
        }
    });

    // Query all data
    selectAllData();

    // Update data
    db.run(UPDATE_DATA, {$deskripsi: filmUpdate.deskripsi, $id: filmUpdate.id}, functi
on(err){
        if(err) {
            console.log(err);
        } else {
            console.log('UDATE DATA');
        }
    });

    selectAllData();

    db.run('DELETE FROM fdb WHERE id=$id', {$id:1}, function(err){
        if(err) {
            console.log(err)
        } else {
            console.log("DELETE DATA");
        };

    })

});
```

```
function selectAllData() {
    db.each(SELECT_DATA, function(err, rows) {
        if (!err) {
            console.log(rows);
        }
    })
}
```

Aplikasi diatas akan membuat tabel `fdb`, memasukkan data baru kemudian data tersebut akan di update dan terakhir akan dihapus.

Contoh diatas memakai API berikut ini

`db.serialize()`

`db.run(operasi_sqlite, callback)`

Dengan memakai fungsi `db.serialize()` maka eksekusi sql akan di eksekusi secara serial atau berurutan dan operasi SQL apa saja bisa di eksekusi oleh node-sqlite3 dengan memakai metode `db.run()` tersebut.

Penjelasan API lebih lanjut untuk Node SQLite bisa dilihat pada link berikut

https://github.com/mapbox/node-sqlite3/wiki/API.

Enkripsi

Kalau anda memakai SQLite dan membutuhkan supaya database ini terenkripsi maka beruntunglah karena node `sqlite3` mendukung penggunaan database SQLite yang terenkripsi.

Enkripsi yang didukung oleh SQLite yaitu melalui ekstensi `sqlcipher`. Sqlcipher merupakan ekstensi sqlite yang mendukung enkripsi 256 bit AES. Ekstensi ini bisa di didownload melalui website berikut

https://github.com/sqlcipher/sqlcipher

sqlcipher

Instalasi ekstensi ini sangat mudah yaitu jika anda berada dalam lingkungan Linux

```
$ sudo apt-get install libsqlcipher-dev
```

Kemudian kompilasi ulang node-sqlite3 dengan perintah berikut ini

```
$ npm install sqlite3 --build-from-source --sqlite_libname=sqlcipher \
--sqlite=/usr/
```

Untuk menggunakan fitur enkripsi ini peru ditambahkan beberapa baris kode berikut pada aplikasi node-sqlite3.

```
//encrypt database
db.run('PRAGMA key="passwordmu!"');
```

Pada contoh CRUD node-sqlite3 pada bagian sebelumnya kode diatas bisa dituliskan sebelum operasi CRUD atau pada saat inisialisasi.

```
...
// Run SQL one at a time
db.serialize(function() {

    //encrypt database
    db.run('PRAGMA key="passwordmu!"');

    // Create table
    db.run(CREATE_TABLE, function(err) {
        if (err) {
            console.log(err);
        } else {
            console.log('CREATE TABLE');
        }
    });
...
```

Bandingkan jika SQLite tidak memakai enkripsi, anda bisa menggunakan SQLite Browser atau tool `hexdump` pada Linux

```
➜ hello-node-sqlite3  hexdump -C film.db
00000000  53 51 4c 69 74 65 20 66  6f 72 6d 61 74 20 33 00  |SQLite format 3.|
00000010  04 00 01 01 00 40 20 20  00 00 00 04 00 00 00 03  |.....@  ........|
00000020  00 00 00 00 00 00 00 00  00 00 00 01 00 00 00 04  |................|
00000030  00 00 00 00 00 00 00 00  00 00 00 01 00 00 00 00  |................|
00000040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 04  |................|
00000060  00 1e 8c 51 0d 00 00 00  02 03 18 00 03 6a 03 18  |...Q.........j..|
00000070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000310  00 00 00 00 00 00 00 00  50 02 06 17 2b 2b 01 59  |........P...++.Y|
00000320  74 61 62 6c 65 73 71 6c  69 74 65 5f 73 65 71 75  |tablesqlite_sequ|
00000330  65 6e 63 65 73 71 6c 69  74 65 5f 73 65 71 75 65  |encesqlite_seque|
00000340  6e 63 65 03 43 52 45 41  54 45 20 54 41 42 4c 45  |nce.CREATE TABLE|
00000350  20 73 71 6c 69 74 65 5f  73 65 71 75 65 6e 63 65  | sqlite_sequence|
00000360  28 6e 61 6d 65 2c 73 65  71 29 81 13 01 07 17 13  |(name,seq).....|
00000370  13 01 82 0d 74 61 62 6c  65 66 64 62 66 64 62 02  |....tablefdbfdb.|
00000380  43 52 45 41 54 45 20 54  41 42 4c 45 20 66 64 62  |CREATE TABLE fdb|
00000390  20 28 20 69 64 20 49 4e  54 45 47 45 52 20 50 52  | ( id INTEGER PR|
000003a0  49 4d 41 52 59 20 4b 45  59 20 41 55 54 4f 49 4e  |IMARY KEY AUTOIN|
000003b0  43 52 45 4d 45 4e 54 2c  20 6a 75 64 75 6c 20 54  |CREMENT, judul T|
000003c0  45 58 54 20 4e 4f 54 20  4e 55 4c 4c 2c 20 72 65  |EXT NOT NULL, re|
000003d0  6c 65 61 73 65 20 54 45  58 54 20 4e 4f 54 20 4e  |lease TEXT NOT N|
000003e0  55 4c 4c 2c 20 69 6d 64  62 20 54 45 58 54 2c 20  |ULL, imdb TEXT, |
000003f0  64 65 73 6b 72 69 70 73  69 20 54 45 58 54 20 29  |deskripsi TEXT )|
00000400  0d 00 00 00 00 04 00 00  03 ac 00 00 00 00 00 00  |................|
00000410  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
```

dan jika SQLite memakai enkripsi bisa dilihat dari screenshot dibawah ini bahwa isi dari database menjadi "meaningless".

```
➜ hello-node-sqlite3  hexdump -C film.db
00000000  45 fa 3c 65 f3 9a dc ca  a3 dc 16 c7 90 64 35 9f  |E.<e.........d5.|
00000010  3e 29 96 a6 14 f1 a4 e3  7c 34 15 48 90 d5 b7 a3  |>).....|4.H....|
00000020  8d 71 d8 ea 03 15 2f d8  3a 58 36 34 22 03 a2 56  |.q..../.:X64"..V|
00000030  3a 1e a5 9c a5 69 5f 55  34 52 89 e1 da 5b 10 d0  |:....i_U4R...[..|
00000040  27 d6 ff 62 65 96 7d 37  f6 0e 42 6d b0 11 f0 fd  |'..be.}7..Bm....|
00000050  1d 80 06 f3 9b df 5c 4e  a3 99 f1 11 22 d6 04 71  |......\N...."..q|
00000060  bc 6b 3d 0e a8 de cc 24  f2 68 fa 14 9a df ea bf  |.k=....$.h......|
00000070  1e 9c 6e 30 4a f8 dc 77  e6 c6 3e 34 6d 96 94 2e  |..n0J..w..>4m...|
00000080  d9 ac 74 5d c8 2a 13 8b  b7 8d 25 41 25 39 22 ec  |..t].*....%A%9".|
00000090  60 f2 0a 2a e9 3e f0 cb  f5 6f f1 ba ab 66 20 c7  |`..*.>...o...f .|
000000a0  30 dd fa b7 4a bc 45 d5  f9 61 19 c9 57 cd a1 41  |0...J.E..a..W..A|
000000b0  81 b0 d7 c9 23 ec 63 a7  ad 3c da a4 a1 55 a2 3c  |....#.c..<...U.<|
000000c0  51 7f db 70 79 6c ba 07  c8 c4 2b 17 f1 f0 a5 eb  |Q..pyl....+.....|
000000d0  5a 6a 6b 70 2f 87 16 f5  27 02 c5 f9 4f 12 5e 2d  |Zjkp/...'...O.^-|
000000e0  71 04 d6 bf b4 f0 bf af  77 15 ef 1a ee 96 f6 5d  |q.......w......]|
000000f0  73 21 fa 5e a5 6b 59 b5  ec 0d 19 0f 06 7d ee b0  |s!.^.kY......}..|
00000100  bd ad bb 2e 82 ac c5 f5  1c 5a ea b9 21 85 d2 82  |.........Z..!...|
00000110  ee f6 c8 60 53 2d 40 4f  b0 15 f6 2e 75 5b 8d 0b  |...`S-@O....u[..|
00000120  92 81 48 a5 d8 a7 8f d2  54 75 d5 e4 0f 90 b8 1e  |..H.....Tu......|
00000130  8f ef 2e 23 93 4e c9 87  f7 3f 46 a6 13 0e e8 67  |...#.N...?F....g|
00000140  94 26 f5 bd 59 36 7e c7  25 0b 8d 9a e6 65 f7 d3  |.&..Y6~.%....e..|
00000150  48 e2 48 5e 2b 91 86 eb  ef 09 24 c7 23 74 21 cd  |H.H^+.....$.#t!.|
00000160  b9 2d fc e2 50 a1 0c 2e  25 d1 15 24 24 57 30 9e  |.-..P...%..$$W0.|
```

MySQL

Kalau anda terbiasa dengan bahasa pemrograman PHP maka pasti sudah tidak asing lagi dengan database relasional yang paling banyak di pakai saat ini yaitu MySQL.

Node MySQL

Implementasi driver dari MySQL untuk platform Node.jssudah sangat mature seperti modul **node-mysql** di https://github.com/felixge/node-mysql/

Instalasi

Seperti biasa untuk menginstall modul ini dan secara otomatis menyimpan nama & versi modul di file `package.json` yaitu dengan memakai perintah berikut

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

Koneksi Sederhana

Untuk membuat koneksi ke database MySQL salah satu caranya adalah dengan menggunakan metode `connection.connect()`

```
//app.js
var mysql = require('mysql');

/**
* Setting opsi dari connection,
* lihat https://github.com/felixge/node-mysql/
*/
var connection = mysql.createConnection({
    host: 'localhost',
    user: 'root',
    password: ''
});

//Membuka koneksi ke database MySQL
connection.connect(function(err){
    if(err) {
        console.log(err);
    } else {
        console.log('Koneksi dengan id '+ connection.threadId);
    }
});

// Query bisa dilakukan di sini

//Menutup koneksi
connection.end(function(err){
   if(err) {
       console.log(err);
    } else {
       console.log('koneksi ditutup!');
   }
});
```

Query

Istilah `query` mungkin menurut mindset umum artinya adalah mengambil data dari database tetapi dalam modul node-mysql ini untuk membuat database atau schema juga didefiniskan sebagai `query`. Ada beberapa bentuk fungsi untuk wrapper `query`

`connection.query(sqlString, callback)`

Dengan memakai statemen SQL kita bisa membuat schema database **ebook** seperti berikut

```
var create_db = 'CREATE DATABASE IF NOT EXISTS ebook'

connection.query(create_db, function(err, result){
    if(err){
      console.log(err);
    } else {
      console.log(result);
    }
)
```

`connection.query(sqlString, value, callback)`

Misalnya untuk menyimpan data semua judul buku pada database **ebook**

```
var ebook = {
    id: 1,
    title: 'Wiro Sableng Pendekar Kapak Maut Naga Geni 212 : Batu Tujuh Warna',
    pengarang: 'Bastian Tito'
}

var insert_sql = 'INSERT INTO ebook SET ?';

connection.query(insert_sql, ebook, function(err, result){
    err ? console.log(err): console.log(result);
})
```

Kalau dibutuhkan `escaping` value sebelum dimasukkan ke database MySQL bisa memakai metode `.escape()` atau pake placeholder `?`.

```
var ebook = {
    id: 1,
    title: 'Wiro Sableng Pendekar Kapak Maut Naga Geni 212 : Batu Tujuh Warna',
    pengarang: 'Bastian Tito'
}

var insert_sql = 'UPDATE ebook SET title = ? WHERE id = ?';

connection.query(insert_sql, [ebook.title, ebook.id], function(err, result){
    err ? console.log(err): console.log(result);
})
```

`connection.query(options, callback)`

Bentuk wrapper `query` yang terakhir ini cukup ringkas. Sesuaikan saja mana bentuk wrapper yang sesuai dengan kebutuhan anda.

```
var ebook = {
    sql: 'INSERT INTO ebook SET pengarang = ?',
    timeout: 14000,
    values: ['Pramoedya Ananta Toer']
}

connection.query(ebook, function(err, result, fields){
    if(err) {
        console.log(err);
    } else {
        console.log(result);
    }
})
```

Metode `query` ini sangat fleksibel dan pada intinya kita bisa memakai statemen SQL yang biasa dipakai untuk query data di MySQL. Jadi penggunaan dari modul npm ini sebenarnya cukuplah mudah.

Untuk penggunaan modul npm ini lebih lanjut silahkan kunjungi Github node-mysql dan untuk contoh aplikasi yang memanfaatkan database ini bisa dilihat pada bab **Pengubah Gambar PNG Ke Data URI**.

MongoDB

Kalau anda sudah terbiasa memakai database relasional seperti MySQL mungkin diperlukan sedikit perubahan *mindset* untuk mengenal tipe database yang namanya **NoSQL**. Seperti arti dari namanya, database ini merupakan database yang tidak memakai bahasa SQL query data tapi bisa secara langsung menggunakan bahasa pemrograman client sebagai contoh adalah MongoDB.

Database **NoSQL** seperti MongoDB menyimpan data sebagai dokumen yang *schema-less* yang artinya yaitu data yang disimpan mempunyai *key - value* yang tidak terikat atau bebas. Anda bisa membayangkannya sebagai data JSON yang tersimpan di database.

Klien bisa berinteraksi langsung dengan database MongoDB dengan menggunakan shell JavaScript `mongo` untuk administrasi dan query data. Sebagai contoh jika kita ingin membuat sample data sebanyak 100 di **collection** `sample` maka perintah dalam shell adalah seperti berikut,

```
$ mongo
MongoDB shell version: 2.6.9
connecting to: test
> use sample;
switched to db sample

> for(var i=0; i<100; i++){ db.sample.insert({band:"Dewa "+i})};
WriteResult({ "nInserted" : 1 })

> db.sample.count()
100

> db.sample.find();
{ "_id" : ObjectId("55cf520e2dd317a13673d11b"), "band" : "Dewa 0" }
{ "_id" : ObjectId("55cf520e2dd317a13673d11c"), "band" : "Dewa 1" }
{ "_id" : ObjectId("55cf520e2dd317a13673d11d"), "band" : "Dewa 2" }
{ "_id" : ObjectId("55cf520e2dd317a13673d11e"), "band" : "Dewa 3" }
{ "_id" : ObjectId("55cf520e2dd317a13673d11f"), "band" : "Dewa 4" }
{ "_id" : ObjectId("55cf520e2dd317a13673d120"), "band" : "Dewa 5" }
{ "_id" : ObjectId("55cf520e2dd317a13673d121"), "band" : "Dewa 6" }
{ "_id" : ObjectId("55cf520e2dd317a13673d122"), "band" : "Dewa 7" }
{ "_id" : ObjectId("55cf520e2dd317a13673d123"), "band" : "Dewa 8" }
{ "_id" : ObjectId("55cf520e2dd317a13673d124"), "band" : "Dewa 9" }
{ "_id" : ObjectId("55cf520e2dd317a13673d125"), "band" : "Dewa 10" }
{ "_id" : ObjectId("55cf520e2dd317a13673d126"), "band" : "Dewa 11" }
{ "_id" : ObjectId("55cf520e2dd317a13673d127"), "band" : "Dewa 12" }
{ "_id" : ObjectId("55cf520e2dd317a13673d128"), "band" : "Dewa 13" }
{ "_id" : ObjectId("55cf520e2dd317a13673d129"), "band" : "Dewa 14" }
{ "_id" : ObjectId("55cf520e2dd317a13673d12a"), "band" : "Dewa 15" }
{ "_id" : ObjectId("55cf520e2dd317a13673d12b"), "band" : "Dewa 16" }
{ "_id" : ObjectId("55cf520e2dd317a13673d12c"), "band" : "Dewa 17" }
{ "_id" : ObjectId("55cf520e2dd317a13673d12d"), "band" : "Dewa 18" }
Type "it" for more
>
```

Untuk lebih jelasnya jika anda tertarik untuk bermain main dengan terminal MongoDB silahkan kunjungi dokumentasinya.

Sebelum kita lanjutkan ke koneksi Node.js dan MongoDB, perlu di ingat beberapa konsep penting dari MongoDB

Collection

Collection adalah kumpulan dari **document**, analoginya walaupun kurang tepat sebenarnya bisa dibilang seperti tabel kalo dalam database relasional.

Document

Document itu sendiri adalah data yang tersimpan di database NoSQL dan didefinisikan oleh yang namanya **Schema**.

Schema

Sebelumnya di katakan bahwa NoSQL menyimpan data yang *schema-less*, memang benar tapi tetap untuk menyimpan suatu dokumen di database biasanya aplikasi bekerja dengan model data tertentu misalnya untuk data `Person` bisa mempunyai *key - value* seperti berikut

```
{
    nama: "Kebo Ijo",
    email: "mbolang@kebo.xyz",
    username: "obek_seloso"
}
```

Node MongoDB

MongoDB menyediakan driver resmi untuk platform Node.js. Module npm ini tersedia di link berikut https://www.npmjs.com/package/mongodb dan dapat dengan mudah di install melalui `npm`

```
$ npm install --save mongodb
```

Penulis mengasumsikan bahwa MongoDB sudah terinstal dan berjalan pada sistem. Untuk memulai koneksi dapat dengan mudah dilakukan seperti script berikut ini,

> app.js

```
var MongoClient = require('mongodb').MongoClient;
var MONGODB_URL = 'mongodb://localhost:27017/sample';

MongoClient.connect(MONGODB_URL, function(err, db){
    err ? console.log(err): console.log('Koneksi ke MongoDB Ok!');
    db.close();
});
```

MongoDB akan membuat database baru jika database tersebut tidak ada, seperti halnya dengan database `sample` pada kode diatas karena sebelumnya database ini tidak ada maka secara otomatis MongoDB akan membuatnya. Bentuk umum URI untuk koneksi ke MongoDB adalah seperti berikut

```
mongodb://<username>:<password>@<localhost>:<port>/<database>
```

Jalankan apikasi di terminal dan jika tidak ada masalah maka akan muncul pesan pada konsol bahwa koneksi ke MongoDB telah sukses.

```
$ node app.js
Koneksi ke MongoDB Ok!
```

Berikutnya akan kita lakukan operasi dasar untuk MongoDB yaitu CRUD tapi sebelumnya kita buat *schema* terlebih dahulu.

> person.js

```
function PersonSchema(data) {
    this.nama = data.nama;
    this.email = data.email;
    this.username = data.username;
};

module.exports = PersonSchema;
```

Schema diatas merupakan model data sederhana yang dituliskan dalam object JavaScript dan tanpa *built-in type casting* ataupun fitur validasi. Jika anda membutuhkan pemodelan data yang lebih handal dan lebih baik, maka pakailah pustaka ODM (Object-Document Modeler) seperti Mongoose.

Driver Node MongoDB menyediakan API yang lengkap untuk bekerja dengan database ini. Silahkan lihat link berikut untuk melihat lebih lengkap tentang API ini

http://mongodb.github.io/node-mongodb-native/2.0/api/Collection.html#insert

Insert

Untuk memasukkan dokumen bisa memakai metode insertOne() untuk memasukkan satu dokumen

> app.js

```
/**
* Balajar Node - MongoDB
*
*
* MIT
* Equan Pr. 2015
*/

var PersonSchema = require('./person.js');
var MongoClient = require('mongodb').MongoClient;
var MONGODB_URL = 'mongodb://localhost:27017/sample';

var person = new PersonSchema({
    nama: 'Kebo Ijo',
    email: 'kebo@ijo.xyz',
    username: 'obek_rebo'
});

MongoClient.connect(MONGODB_URL, function(err, db){
    err ? console.log(err): console.log('Koneksi ke MongoDB Ok!');
    db.collection('persons').insertOne(person, function(err, result){
        if(err){
           console.log(err);
        } else {
           console.log(result);
        }
        db.close();
    })
});
```

Secara otomatis operasi insert ini akan menghasilkan *primary key* `_id` yang unik yaitu berupa `ObjectId`. Yang membedakan `_id` ini dengan id pada database yang lain adalah dengan `ObjectId` bisa didapatkan kapan data ini dimasukkan melalui pemakaian metode `getTimestamp()`.

```
$ mongo
MongoDB shell version: 2.6.9
connecting to: test

>  _id = ObjectId()
ObjectId("55d81f48bb934a51424dbd37")

> ObjectId("55d81f48bb934a51424dbd37").getTimestamp();
ISODate("2015-08-22T07:05:44Z")

>
```

Jika anda mempunyai banyak dokumen, untuk memasukkan dokumen-dokumen tersebut ke collection `persons` anda bisa memakai metode `insertMany()`.

Update

Operasi update data juga cukup mudah apalagi jika anda sangat pahamn tentang MongoDB. Untuk meng-update data bisa dilakukan melalui metode `updateOne()` atau `updateMany()`.

> app.js

```
var PersonSchema = require('./person.js');
var MongoClient = require('mongodb').MongoClient;
var MONGODB_URL = 'mongodb://localhost:27017/sample';

var person = new PersonSchema({
    nama: 'Kebo Ijo',
    email: 'kebo@ijo.xyz',
    username: 'obek_rebo'
});

MongoClient.connect(MONGODB_URL, function(err, db){
    err ? console.log(err): console.log('Koneksi ke MongoDB Ok!');
    db.collection('persons').insertOne(person, function(err, result){
        if(err){
             console.log(err);
        } else {
             console.log('Simpan data person ok!');

           //update data
           var personUpdate = {
               nama: 'Sukat Tandika'
           }
           db.collection('persons').updateOne({nama: person.nama}, personUpdate, funct
ion(err, result){
               if(err) {
                 console.log(err);
               } else {
                 console.log('Data person berhasil dimodifikasi!');
               }
               db.close();
           })
        }
    })
});
```

Metode `updateOne()` mempunyai beberapa argumen seperti berikut

```
updateOne(filter, update, options, callback)
```

Untuk data update anda bisa memakai operator seperti `$set` contohnya seperti berikut ini

```
db.collection('persons').updateOne({nama: person.nama}, {$set:{nama: 'Angel'}}, functi
on(err, result){
    if(err) {
        console.log(err);
    } else {
        console.log('Data person berhasil dimodifikasi!');
    }

    db.close();
})
```

dari beberapa options yang terpenting adalah key `upsert` yaitu *update insert* dan jika option ini diberikan maka jika data yang akan di-update tidak ada maka MongoDB secara otomatis akan membuat data yang baru. Untuk lebih jelasnya anda bisa melihat dokumentasi dari API `updateOne()` .

Query

Query data pada database MongoDB dapat dengan mudah dilakukan dengan memakai metode `find()` , sebagai contoh untuk menemukan data `person` pada collection `persons`

```
MongoClient.connect(MONGODB_URL, function(err, db){
    if(!err){
        findPerson({nama: 'Morbid Angel'}, db, function(err, doc){
            if(!err){
                console.log(doc);
            } else {
                console.log(err);
            }
            db.close();
        });
    } else {
        console.log(err);
    }
});


function findPerson(filter, db, callback){
  var cursor = db.collection('persons').find(filter);
  cursor.each(function(err, docResult){
      if(err){
          callback(err, null);
      } else {
          if(docResult != null) {
              console.log(docResult);
              callback(null, docResult);
          } else {
              callback(null, 'empty!');
        }
      }
  })
}
```

Dengan metode `find()` anda bisa memakai operator query seperti `$lt` , `$gt` , operator kondisi `AND` , `OR` dll. Untuk lebih lengkapnya silahkan lihat dokumentasi query dari driver node MongoDB.

Delete

Untuk menghapus data anda bisa menggunakan fungsi `deleteOne()` atau `deleteMany()` . Misalnya untuk menghapus semua data pada collection `persons` anda bisa menggunakan empty object `{}` sebagai query.

```
function deleteAllPerson(db, callback){
    db.collection('persons').deleteMany({}, function(err, rec){
        if(!err) {
            callback(null, rec.result.n);
        } else {
            callback(err, null);
        }
    })
}
```

atau jika ingin menhapus satu data pada collection `person`

```
fucntion deletePerson(filter, db, callback){
    db.collection('persons').deleteOne(filter, function(err, rec){
        if(!err) {
            callback(null, rec.result.n);
        } else {
            callback(err, null);
        }
    })
}
```

Dua fungsi ini bisa anda lihat secara lengkap pada dokumentasi API MongoDB.

Mongoose

Seperti dikatakan sebelumnya untuk memudahkan pemodelan data di MongoDB anda bisa memakai pustaka Mongoose. Meskipun sebenarnya Mongoose memakai objek JavaScript biasa dan mendukung banyak metode `query` data tetapi yang paling membedakan adalah Mongoose mendukung schema type dan validasi di level schema.

```
data:{type: schemaType}
```

Schema Type

Ada beberapa schema type yang didukung oleh pustaka ini yaitu

- String
- Number
- Date
- Boolean
- Buffer (untuk data binary misalnya gambar, mp3 dll.)
- Mixed (data dengan tipe apa saja)
- Array
- ObjectId

Pemodelan Data

Lalu bagaimana Mongoose memodel data? ambil contoh misalnya dalam pengembangan aplikasi, data yang akan dipakai seperti berikut

```
var lokasiWisataKotaBatu = [{
  nama: 'Batu Night Square',
  alamat: 'Jl. Oro oro ombo 13, Tlekung'
  rating: 3,
  fasilitas: ['Wahana bermain','Roller coaster', 'Taman lampion']
}]
```

Dari data diatas dengan memakai Mongoose sangat mudah untuk diterjemahkan ke skema dan tipe data untuk tiap field

```
var lokasiWisataKotaBatuSchema = new mongoose.schema({
  nama: String,
  alamat: String,
  rating: Number,
  fasilitas: [String]
})
```

Bagaimana dengan validasi data?. Jika anda lihat field data `rating` diatas dan umumnya mempunyai nilai antara 0 - 5 dan jika secara default diberi nilai 0 maka skema `rating` akan menjadi

```
rating: {type: Number, 'default': 0}
```

Anda juga bisa memakai key `required` untuk menandakan bahwa field data tersebut bersifat wajib ada misalnya

```
nama: {type: String, required: true}
```

Untuk lebih jelasnya silahakan lihat dokumentasi dari Validators Mongoose.

Aplikasi node.js dengan database MongoDB yang memakai pustaka Mongoose dapat dikatakan mempunyai hubungan relasi satu-satu artinya dengan memakai model Mongoose berarti secara langsung aplikasi akan memakai data pada MongoDB.

Kalau misalnya anda bekerja dengan database relasional MySQL maka anda akan memakai bahasa SQL seperti `INSERT INTO table VALUES (field1=data)` untuk memasukkan data sedangkan seperti anda ketahui jika dengan database NoSQL untuk menyimpan model object cukup dengan memakai metode misalnya `model.save()` . Dalam Mongoose untuk memakai model yang telah mempunyai schema seperti diatas

```
var LokasiModel = mongoose.model('Lokasi', lokasiWisataKotaBatuSchema, 'lokasiWisataBa
tu')
```

Kode diatas memberi arti bahwa aplikasi akan memakai model `Lokasi` dengan **schema type** yang telah di definisikan sebelumnya yaitu `lokasiWisataKotaBatuSchema` dan model ini akan di simpan pada koleksi data di MongoDB dengan nama koleksi adalah `lokasiWisataBatu` .

Untuk memakai model `Lokasi` caranya adalah dengan menciptakan `instance` dari model tersebut

```
var lokasiModel = new LokasiModel()
lokasiModel.nama = 'Museum Angkut'
lokasiModel.alamat = 'Jl. Kembar 11'
lokasiModel.rating = 5
lokasiModel.fasilitas = ['Restoran', 'Parkir luas']
```

Kemudian untuk menyimpan data tersebut ke dalam database MongoDB

```
lokasiModel.save(function(err){
  if(!err) console.log('Data Disimpan!')
})
```

Cukup mudah bukan dan memang pustaka ini bertujuan untuk memudahkan developer dalam pemodelan data terutama jika data yang akan dimodel mempunyai struktur yang rumit misalnya nested document. Untuk lebih lengkapnya silahkan anda mengunjungi situs resmi mongoosejs.com.

Projek

Bagi anda yang sudah mengerti gambaran umum dari pustaka Mongoose ini silahkan mengutak atik contoh projek sederhana yang memakai framework ExpressJS dan Mongoose, Person REST API.

Testing

Perangkat lunak tidak ada yang sempurna dan pasti mempunyai *bugs*. Salah satu cara untuk menguranginya atau mencegahnya untuk muncul di kemudian hari adalah dengan jalan melakukan pengetesan. Pengetesan memberikan keyakinan pada developer bahwa perangkat lunak bekerja sebagai mana mestinya dan jika ada perubahan misalnya penambahan atau pengurangan fitur dipastikan bahwa pengetesan akan selalu berhasil.

Ada banyak tipe pengetesan tetapi dalam pengembangan aplikasi web ada dua tipe pengetesan yang penting untuk diketahui yaitu **Unit Testing** dan **Acceptance Testing**.

Unit Testing

Unit Testing dilakukan langsung pada level kode aplikasi yaitu pada fungsi atau metode dan ada dua metodologi yang sering digunakan yaitu **TDD** dan **BDD**. Dalam JavaScript tersedia banyak pustaka yang dibuat untuk pengetesan seperti module `assert` bawaan dari Node.js, Nodeunit, Mocha, Jasmine dll.

Test Driven Development (TDD)

Istilah TDD ini menjadi sangat keren beberapa tahun belakangan ini. Konsep TDD adalah melakukan pengetesan terlebih dahulu baru kemudian menulis kode dari aplikasi, jadi mindset dari pembuatan perangkat lunak dibalik kalo dalam pengembangan perangkat lunak yang konvensional alurnya yaitu menulis kode terlebih dahulu baru kemudian menulis kode untuk pengetesan sedangkan dalam **Test Driven Development** proses ini dibalik.

Kelemahan pengetesan ini adalah pada proses awal biasanya dibutuhkan waktu yang relatif lebih lama untuk mengembangkan aplikasi karena harus menulis kode pengetesan terlebih dahulu tetapi keuntungan jangka panjangnya yaitu aplikasi yang dihasilkan biasanya mempunyai *bugs* yang lebih sedikit.

Behavior Driven Development (BDD)

Pengetesan secara BDD memakai bahasa atau metode pengetesan yang lebih ramah yaitu dengan melakukan pengetesan dengan cara atau alur seperti **user story** sehingga pengetesan ini relatif lebih populer karena secara bahasa menjembatani atau memungkinkan kolaborasi antara developer dan klien. Ada banyak pustaka pengetesan yang memakai BDD seperti Mocha, Jasmin, Vows dll. Sebagai gambaran contoh untuk pustaka Mocha misalnya

```
describe('Upload file', function(){
  it('Harus bisa menerima file jpeg', function(){
    //testing disini
  })

  it('Harus bisa menerima file png', function(){
    //testing disini
  })

  it('Harus bisa menerima file xcf', function(){
    //testing disini
  })
})
```

Kalau diterjemahkan secara bahasa manusia maka kode testing BDD diatas akan mudah dibaca yaitu *Upload file harus bisa menerima file jpeg, png atau xcf*.

Acceptance Testing

Untuk pengetesan dari antar muka aplikasi web beserta fungsinya contohnya seperti pengetesan penekanan button apakah berfungsi dengan baik atau tidak adalah termasuk dari **Acceptance Testing**. Testing ini bisa dilakukan dengan memakai bantuan browser dan dengan munculnya *headless browser* seperti PhantomJS memungkinkan pengetesan dilakukan secara terintegrasi dengan kode *back-end*.

> Catatan penting dari testing adalah pengetesan diusahakan berjalan secara automatis sehingga jika ada kesalahan yang terjadi maka developer bisa segera memperbaikinya, maka dari itu testing pasti mempunyai yang namanya info laporan entah itu berupa file, output terminal atau antarmuka berupa web. Pada proses pengembangan perangkat lunak modern, automasi testing merupakan bagian yang sangat penting.

Dalam buku ini pengetesan akan dibatasi pada pengetesan aplikasi web khususnya aplikasi Node.js dengan REST dengan memakai Mocha.

Pengetesan REST

Kita ambil contoh server yang dibuat dengan ExpressJS. Server dibawah ini akan mengeluarkan data berupa JSON pada URL `/`.

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

app.get('/', function (req, res) {
  res.json({
    message: 'hello'
  })
})

module.exports = app
```

Lalu bagaimana cara mengetest URL `/` diatas? dengan memakai Mocha, module `assert` dan modul Supertest yaitu modul npm yang khusus untuk mengetest server HTTP. Mocha termasuk pustaka pengetesan yang bisa dipakai secara BDD ataupun TDD. Pustaka ini secara default memakai style BDD dan metode yang yang biasa dipakai adalah `describe` & `it`.

```
var mocha = require('mocha')
var request = require('supertest')
var assert = require('assert')
var app = require('../app')

describe('Test REST API', function(){
  describe('GET /', function(){
    it('should return json with key "message" and value "hello"', function(done){
      request(app).get('/')
      .set('Accept', 'application/json')
      .expect('Content-Type', /json/)
      .expect(200)
      .expect(function(res){
          assert.equal(res.body.message, 'hello')
      })
      .end(function(err, res){
          if(err) return done(err)
          done()
      })
    })
  })
})
```

Perlu anda ingat bahwa supertest akan secara otomatis menjalankan server ExpressJS dan mengalokasikan port sehingga anda tidak perlu mejalankan server secara langsung.

```
request(app)
```

Kalau diterjemahkan dalam bahasa manusia kode diatas akan mengetest beberapa kondisi yaitu

```
expect('Content-Type', /json/)
```

kode diatas artinya diharapkan response dari GET untuk URL `/` adalah bertipe `json`

```
expect(200)
```

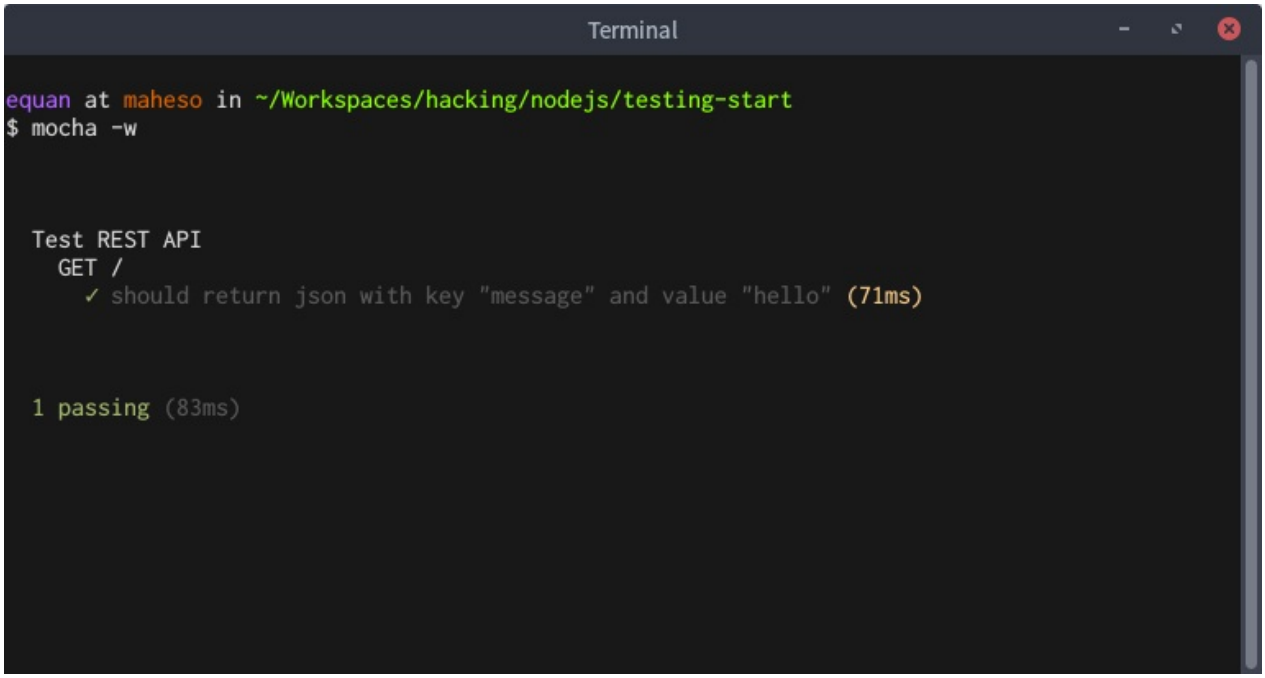
kode diatas artinya **response code** dari request HTTP harus mempunyai kode 200.

```
expect(function(res){
  assert.equal(res.body.message, 'hello')
})
```

dan kode diatas artinya diharapkan bahwa **response body** dengan field `message` mempunyai value `hello` dan tentu saja anda bisa menambahkan banyak kondisi untuk pengetesan yang lebih detil.

Untuk melakukan pengetesan cukup dengan menginstal dan jalankan `mocha` maka secara otomatis mocha akan menjalankan berkas berkas pengetesan yang berada pada folder `test`.

```
$ npm install -g mocha
```



Sebenarnya banyak yang bisa dijelaskan tentang pengetesan dan sudah banyak resource online diluar sana yang membahas tentang ini tapi yang perlu anda ingat adalah pastikan pengetesan dimasukkan dalam alur kerja dalam pengembangan perangkat lunak yang sedang anda kerjakan.

Automasi

Dalam prakteknya biasanya testing ini dilakukan secara otomatis misalnya jika anda memakai Github dalam pengembangan perangkat lunak open source maka anda bisa memakai server **Travis** yang dengan setting konfigurasi tertentu akan menjalankan pengetesan jika anda menge-*push* kode ke repository Github atau anda bisa memakai *build server* yang ada diluaran seperti **Jenkins**, **Bamboo**, **TeamCity** dll.

Travis

Jika anda memakai Github dan menggunanakan Travis maka secara default server travis ini akan menjalankan command

```
$ npm test
```

sehingga dalam package.json yang anda push ke Github anda perlu menulis script untuk pengetesan misalnya jika memakai Mocha

```
"scripts": {
    "test": "mocha"
}
```

.travis.yml

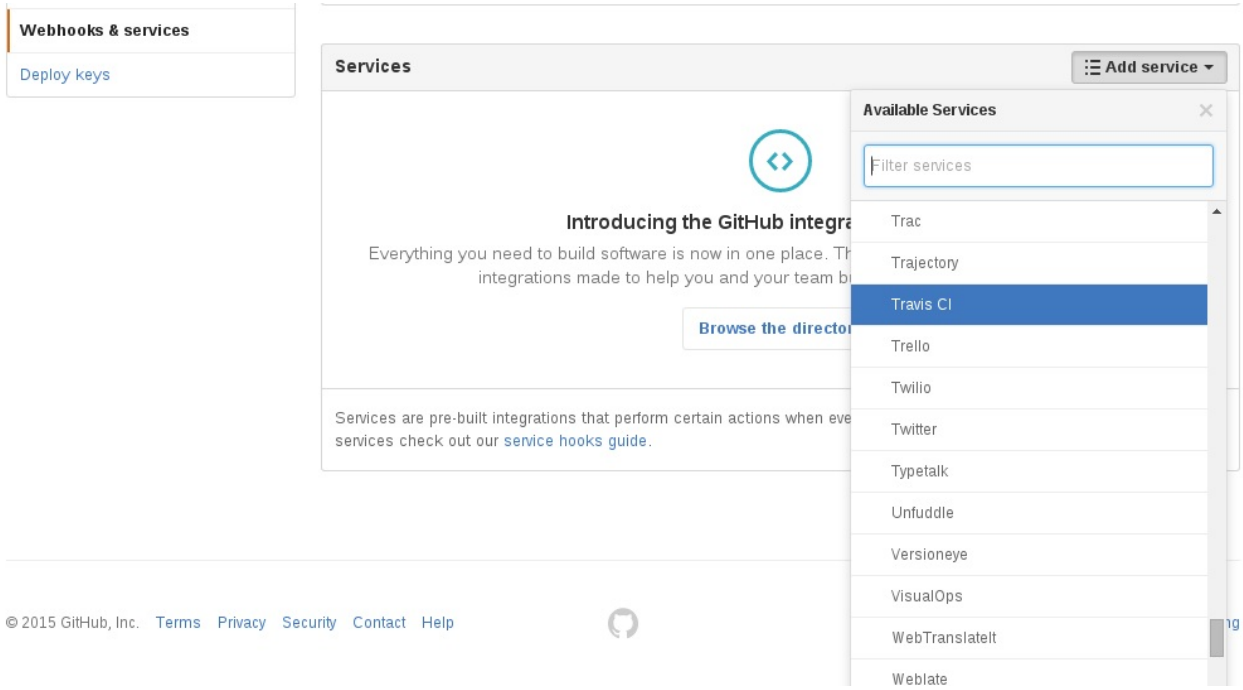
Untuk setting Travis anda cukup membuat file **.travis.yml** pada folder projek misalnya seperti dibawah ini (contoh source code projek test ada di sini)

```
.
├── app.js
├── .travis.yml
├── node_modules
├── package.json
└── test
    └── app.test.js
```

Dengan isi file tersebut adalah versi dari Node.js dimana test akan dijalankan. Misalnya untuk menjalankan pengetesan Node.js versi 4.1 dan 4.0

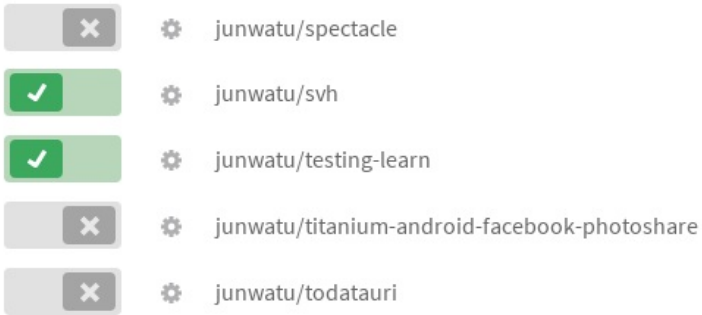
```
language: node_js
node_js:
  - "4.1"
  - "4.0"
```

Kemudian anda perlu mengatur konfigurasi repo di Github (Testing Learn). Supaya Github bisa memakai service dari Travis maka anda perlu meengubah konfigurasi repo (Masuk ke menu **Settings**)



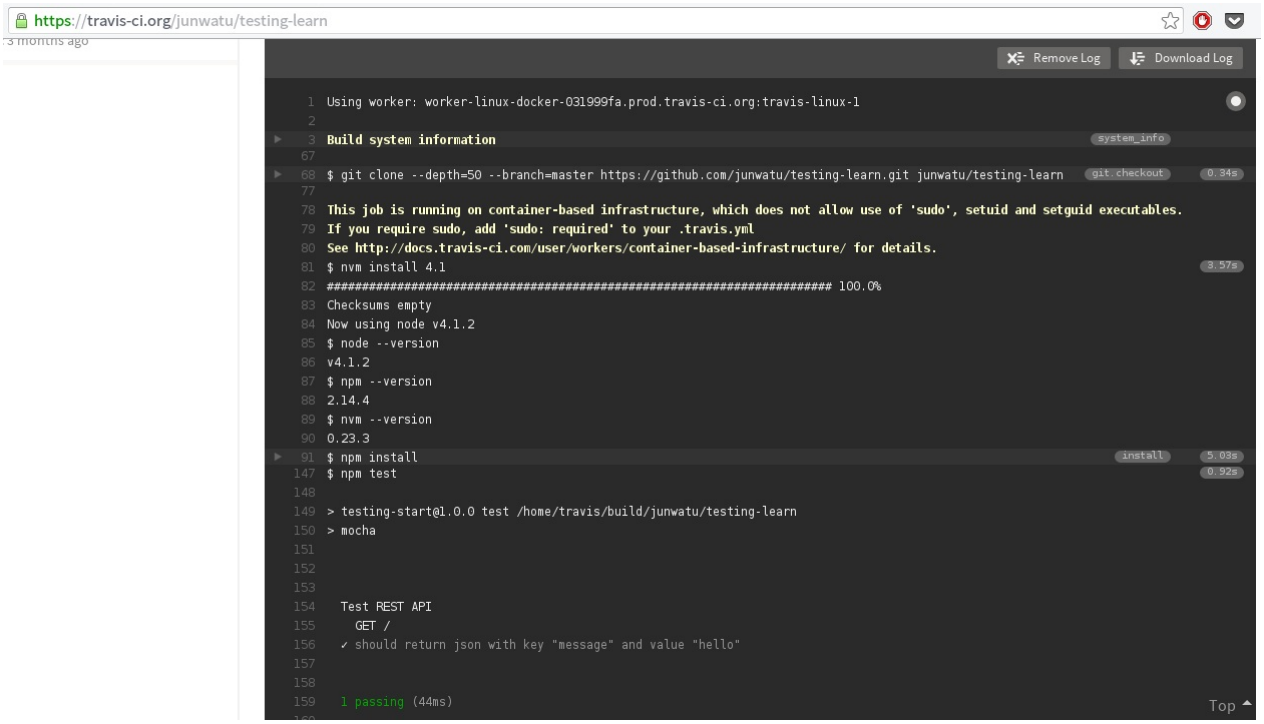
Enable Build

Supaya build selalu dijalankan oleh Travis maka anda harus meng-**ON**-kan tombol enable di travis-ci.org pada repo yang anda inginkan.



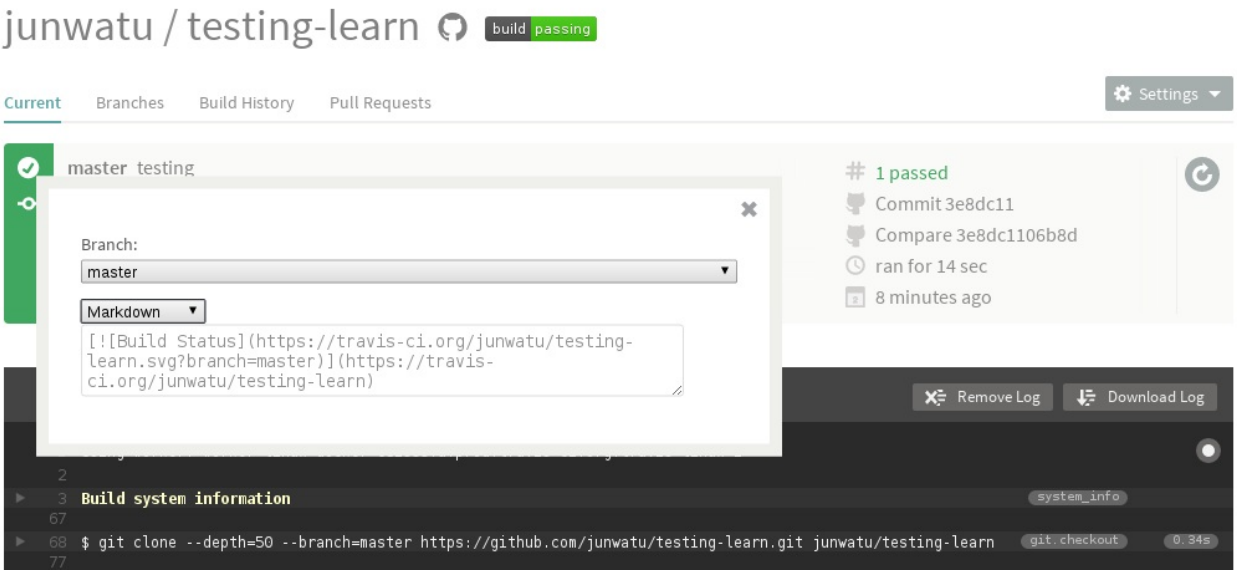
Test Build

Build akan dijalankan oleh Travis setiap ada kode baru di Github atau anda bisa menggunakan tombol **Test Service** pada **Settings** -> **Webhooks & services** untuk **force build**.

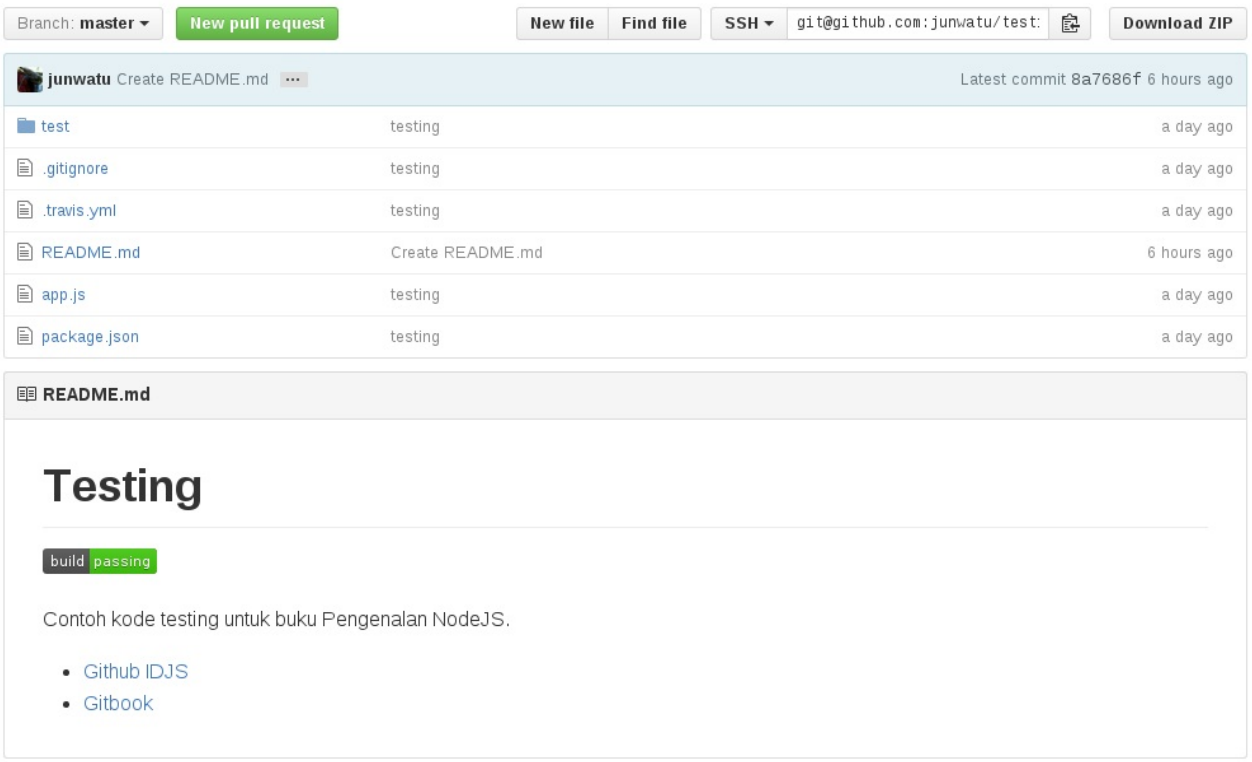


Notifikasi

Notifikasi Travis bisa di integrasikan dengan Slack, Email, dll atau biasanya dengan gambar status di README repo. Untuk konfigurasi gambar status klik tombol **Build/Passing** di sebelah nama repo di Travis dan *copy paste* ke README repo di Github.

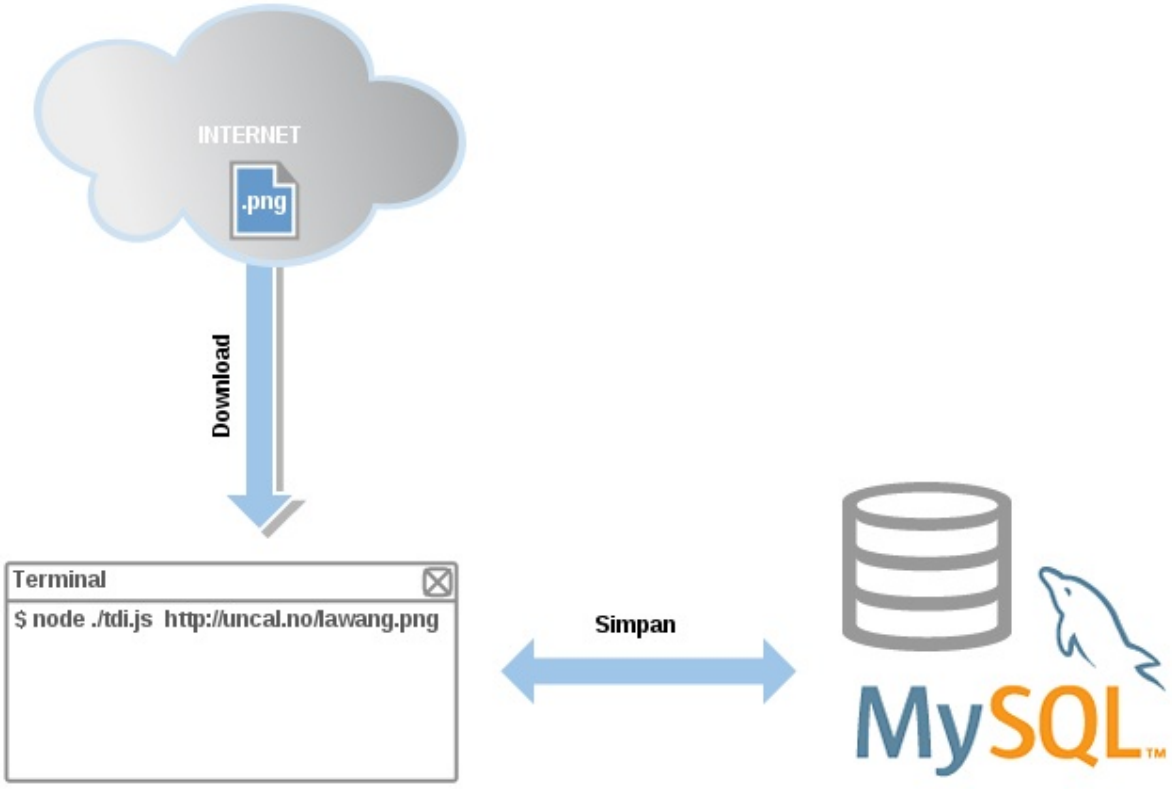


Sehingga jika anda membuka repo di Github maka akan muncul gambar status yang apakah build sukses atau dalam status error.


Branch: master
New pull request
New file
Find file
SSH
git@github.com:junwatu/test:
Download ZIP
junwatu Create README.md
Latest commit 8a7686f 6 hours ago
test testing a day ago
.gitignore testing a day ago
.travis.yml testing a day ago
README.md Create README.md 6 hours ago
app.js testing a day ago
package.json testing a day ago
README.md
Testing
build passing
Contoh kode testing untuk buku Pengenalan NodeJS.
Github IDJS
Gitbook

ToDataURI

Aplikasi ini adalah aplikasi Node.js CLI yang mengubah gambar png ke format data URI dan kemudian menyimpan data tersebut ke database MySQL. Proses ini bisa digambarkan seperti gambar alur dibawah ini.



Data URI

Bagi yang belum tahu Data URI adalah salah satu cara untuk meng-embed data secara inline alih-alih harus mengambil data tersebut dari resource luar. Data yang akan diubah ke Data URI disini bisa berupa image, file csv dll. Untuk aplikasi ini hanya dibatasi untuk gambar berformat png.

Format Data URI adalah seperti berikut ini

```
data:[<MIME-type>][;charset=<encoding>][;base64],<data>
```

Umumnya untuk gambar jenis encoding yang dipakai adalah `base64`. Sebagai contohnya lihat format gambar png yang telah dirubah ke Data URI berikut ini

```

qgAAACBjSFJNAABtmAAAc44AAPXqAACD0AAAfHIAAN7iAAAyWgAAI0zujJoVAAADAFBMVEXW1tYxLS6EgYKtrK
y7uro/OzxoZWZMSUrk5OR2c3SfnZ7y8vJaV1iRj5DJyMgjHyD///8RERESEhITExMUFBQVFRUWFhYXFxcYGBgZ
GRkaGhobGxscHBwdHR0eHh4fHx8gICAhISEiIiIjIyMkJCQlJSUmJiYnJycoKCgpKSkqKiorKyssLCwtLS0uLi
4vLy8wMDAxMTEyMjIzMzM0NDQ1NTU2NjY3Nzc4ODg5OTk6Ojo7Ozs8PDw9PT0+Pj4/Pz9AQEBBQUFCQkJDQ0NE
RERFRUVGRkZHR0dISEhJSUlKSkpLS0tMTExNTU1OTk5PT09QUFBRUVFSUlJTU1NUVFRVVVVWVlZXV1dYWFhZWV
laWlpbW1tcXFxdXV1eXl5fX19gYGBhYWFiYmJjY2NkZGRlZWVmZmZnZ2doaGhpaWlqampra2tsbGxtbW1ubm5v
b29wcHBxcXFycnJzc3N0dHR1dXV2dnZ3d3d4eHh5eXl6enp7e3t8fHx9fX1+fn5/f3+AgICBgYGCgoKDg4OEhI
SFhYWGhoaHh4eIiIiJiYmKioqLi4uMjIyNjY2Ojo6Pj4+QkJCRkZGSkpKTk5OUlJSVlZWWlpaXl5eYmJiZmZma
mpqbm5ucnJydnZ2enp6fn5+goKChoaGioqKjo6OkpKSlpaWmpqanp6eoqKipqamqqqqrq6usrKytra2urq6vr6
+wsLCxsbGysrKzs7O0tLS1tbW2tra3t7e4uLi5ubm6urq7u7u8vLy9vb2+vr6/v7/AwMDBwcHCwsLDw8PExMTF
xcXGxsbHx8fIyMjJycnKysrLy8vMzMzNzc3Ozs7Pz8/Q0NDR0dHS0tLT09PU1NTV1dXW1tbX19fY2NjZ2dna2t
rb29vc3Nzd3d3e3t7f39/g4ODh4eHi4uLj4+Pk5OTl5eXm5ubn5+fo6Ojp6enq6urr6+vs7Ozt7e3u7u7v7+/w
8PDx8fHy8vLz8/P09PT19fX29vb39/f4+Pj5+fn6+vr7+/v8/Pz9/f3+/v7///8EKbe0AAAACXBIWXMAAAsTAA
ALEwEAmpwYAAAB/0lEQVRIx62W2W7sIBBE2Rf3UvX/X5sHYyczyb2eGRnJEjZw6C4KTOD7pQAtXHUKH4AthRab
ldvBIQZmm7GHm8HcOiVTErabweyF6mSJ82bwNjl2Uexe8IiyVxzlVjDnoYEdU9wEFoxVqdut4FFPU9dbwTmdsc
f8Cjj/6BXOwUzP85vxWLXZXwGbSRAnGQZdJZBOuqhzBJLBz34jLvL42xi/wB5T69JTS66uJEZPKXpLSaWntBR1
LdAj5JjlFbCS2BLpC2yJ1C2qxtx6OzwQm9YVqWway4vgolKmq1dxZJVRcx9issmoa0CJW+hnoF7n5eKFRCqzzh
ASTbMya5rBWyssTX+69nt1pfR2j93W4KWAIWq6AAc/8nN/45diVxpPIO6btQMPBpWGvZjMvRKNtojl96H/9K7A
3jcDwM+WhGhmZuYJmGaWgHKAKVblAtwQhWRFBTg8kKS4U890dTlYYfatwbALcInIpAMZoO0MB9jQ3d3/CeaVFG
6opEL9ERzqrmz6GDywP09gMrh7BvxTMCfaROcvMElygZM2eQDnZyP/AQ4AkBe4urtvAIO7uxvgEzGXiB6RDTW7
u7so4iWYCtQ9TNFlXvCopVNtdDnbvdX6X3BwIcWL7B4j5THifV8Gd/eJGMixPouPd84K3/dEfc5zv1yYyac3IV
uZtvfPqi9DYJrPLQNESAAAAABJRU5ErkJggg==
```

Cukup panjang memang kalo di jadikan ke format Data URI. Coba kopi contoh data uri diatas dan *paste* ke browser url dan jangan lupa tekan enter.

Penggunaan

Untuk kode sumber selengkapnya bisa di lihat di Github pada branch `todatauri-mysql` atau ketik perintah berikut pada command line.

```
$ git clone -b todatauri-mysql https://github.com/junwatu/todatauri todatauri-mysql

$ cd todatauri-mysql

$ npm install
```

Ada tiga penggunaan aplikasi ini yaitu

Download PNG & Simpan Ke MySQL

```
$ node ./tdi.js <url_image_png_dari_internet>
```

Aplikasi ini akan mendownload file image png dari internet dan mendukung protokol `http` ataupun `https`.

Demo

```
$ node ./tdi.js
```

Jika dijalankan tanpa argumen maka hasilnya adalah keluaran demo yang tidak lain adalah contoh format Data URI.

Tampilkan Data MySQL

```
$ node ./tdi.js show
```

Jika ada argumen `show` maka data dalam MySQL akan ditampilkan semua. Hati hati jika anda banyak mengkonversi banyak data gambar png.

Konverter PNG Ke Data URI

Mari kita lihat isi dari file `todatauri.js`. Pada dasarnya file ini merupakan pustaka sederhana yang akan mendownload image berformat `.png` di internet dan kemudian mengubahnya menjadi format **Data URI**.

Jadi anda bisa menggunakan pustaka ini seperti halnya modul npm dengan bantuan `require`.

```
/**
 * todatauri.js
 * Download image and convert it to  Data URI
 *
 * WTFPL
 * (c) 2015, Equan Pr.
 */
var fs = require('fs');
var http = require('http');
var https = require('https');
var url = require('url');

function Util() {

    var defaultImage = 'https://upload.wikimedia.org/wikipedia/commons/thumb/d/d9/Node
.js_logo.svg/320px-Node.js_logo.svg.png';
    var imageDest = 'download.png';
    var file;

    return {
        toDataURI: function(urlArg, cb) {
            var imageURL;
            urlArg ? imageURL = urlArg : imageURL = defaultImage;
            if(url.parse(imageURL).protocol === 'https:') {
                Util().httpsGetImage(imageURL, function(err, data){
                    err ? cb(err,null): cb(null, data);
                }) ;
            } else {
                Util().httpGetImage(imageURL, function(err, data){
                    err ? cb(err, null): cb(null, data);
                });
            }
        },

        httpGetImage: function(url, cb) {
            file = fs.createWriteStream(imageDest);

            http.get(url, function(response) {
                response.pipe(file);
```

```
                console.log('Download node.js defaultImage from', defaultImage);
                console.log('\n');

                file.on('finish', function() {
                    file.close();
                    Util().imageToDataUri(imageDest, function(err, data) {
                        err ? cb(err, null) : cb(null,data);
                    });
                });
            })
        },

        httpsGetImage: function(url, cb) {
            file = fs.createWriteStream(imageDest);

            https.get(url, function(response) {
                response.pipe(file);

                console.log('Download node.js defaultImage from', defaultImage);
                console.log('\n');

                file.on('finish', function() {
                    file.close();
                    Util().imageToDataUri(imageDest, function(err, data) {
                        err ? cb(err, null) : cb(null,data);
                    });
                });
            })
        },

        imageToDataUri: function(imageName, cb) {
            //TODO: lookup mime type here
            var mime = 'image/png';
            var encoding = 'base64';

            var uri = 'data:' + mime + ';' + encoding + ',';

            fs.readFile(imageName, function(err, buf) {
                if (err) return cb(err, null);
                cb(null, uri + buf.toString(encoding));
            })
        }
    }
}

module.exports = Util();
```

Fungsi utama pada file `todatauri.js` yaitu fungsi `imageToDataUri()` dan jika dilihat isinya maka sebenarnya sangat sederhana untuk mengubah gambar png ke encoding `base64` yang akan digunakan di Data URI.

```
buf.toString(encoding))
```

`fs.readFile(filename[, options], callback)` merupakan fungsi untuk membaca file secara asinkron dan jika encoding tidak di berikan maka hasil pembacaan file akan mengembalikan data berupa `buffer`. Sehingga untuk mengubah data `buffer` ini ke `base64` cukup dengan memakai metode `.toString(encoding)`.

Koneksi MySQL

Untuk koneksi ke database MySQL cukup mudah seperti yang telah dibahas pada bab sebelumnya tentang Node dan Database MySQL.

```
/**
 * todatauri.js dengan Mysql
 *
 *
 * WTFPL
 * Equan Pr.
 * 2015
 */
var Util = require('./todatauri.js');
var mysql = require('mysql');

var connection = mysql.createConnection({
    host: 'localhost',
    user: 'root',
    password: ''
});

connection.connect(function (err) {
    if (err) {
        console.log(err);
    } else {
        console.log('Koneksi MySQL dengan id ' + connection.threadId);
    }
});

var create_database = 'CREATE DATABASE IF NOT EXISTS data_uri ';
var create_table = "CREATE TABLE IF NOT EXISTS data_uri.images (id INT AUTO_INCREMENT
PRIMARY KEY, data TEXT)";

connection.query(create_database, function (err, res) {
    if (err) {
        console.log(err);
    } else {
        console.log('Create database data_uri [ok]');

        connection.query(create_table, function (err, result) {
            if (err) {
                console.error(err);
            } else {
                console.log('Create table images [ok]');
                main(process);
            }
        })
    }
```

```
});

function insertData(data, connection) {
    connection.query('INSERT INTO data_uri.images SET ?', { data: data }, function (er
r, res) {
        if(err) {
            console.log(err);
        } else {
            console.log(res);
            closeConnection();
        }
    })
};

function showData(connection) {
    connection.query('SELECT * FROM data_uri.images', function(err, result){
        if(err) {
            console.log(err);
        } else {
            console.log('\n\nMySQL Database\n============================\n');
            result.forEach(function(element, index){

                console.log('id \u2192', element.id);
                console.log('image \u2192', element.data);
            });

            closeConnection();
        }
    })
}

function main(process) {
    if (process.argv[2]) {
        if (process.argv[2].toLowerCase().trim() == 'show') {
          showData(connection);
        } else {
            Util.toDataURI(process.argv[2], function (err, data) {
                if (!err) {
                    insertData(data, connection);
                }
            })
        }
    } else {
        Util.toDataURI(null, function (err, data) {
            if (!err) {
                console.log(data);
                closeConnection();
            }
        });
    }
}

function closeConnection(){
```

```
    connection.end(function(err){
       if(err) {
           console.log(err);
        }
    });
}
```

Parsing argumen dilakukan oleh `process` yang merupakan object global di Node.js. Perlu diingat bahwa JavaScript memulai array dengan index 0 sehingga argumen yang diperlukan berada pada index 2

```
process.argv[2]
```

Jika anda berada pada platform UNIX atau GNU LINUX script diatas bisa dirubah untuk `self executable` dengan menambahkan script berikut pada line 1 di file `tdi.js`

```
#!/usr/bin/env node
```

Kemudian untuk menjalankannya cukup mudah seperti berikut

```
$ ./tdi.js
```

Script `tdi.js` sebenarnya cukup sederhana dan cukup untuk melakukan tugas yang di inginkan. Mungkin kelihatan agak rumit karena banyak `callback` di sana sini tetapi kalau anda sudah terbiasa memprogram secara asinkron pasti mudah sekali untuk melihat kode diatas.

Penulis sarankan untuk ke depannya jika diinginkan penulisan tanpa `callback` bisa dipelajari tentang `Promises` JavaScript ES6.

Person REST API

Aplikasi ini merupakan contoh sederhana layanan server melalui **REST** (**Re**presentational **S**tate **T**ransfer) yaitu mekanisme untuk komunikasi dengan server melalui protokol HTTP yang mudah untuk digunakan daripada memakai mekanisme protokol lama sepeti CORBA, SOAP ataupun RPC.

Aplikasi REST memakai metode HTTP seperti POST, PUT, GET dan DELETE untuk menambah, mengubah, mengambil ataupun menghapus resource yang ada pada server. Sehingga dalam pengembangan aplikasi khususnya dengan memakai Node.js, mekanisme REST sangat mudah untuk digunakan.

Susunan API

Berikut susunan API yang akan digunakan dalam contoh aplikasi ini. Contoh API ini mungkin tidak mengikuti *best practice* tapi penulis rasa sudah cukup untuk menggambarkan bagaimana menyusun aplikasi REST secara sederhana.

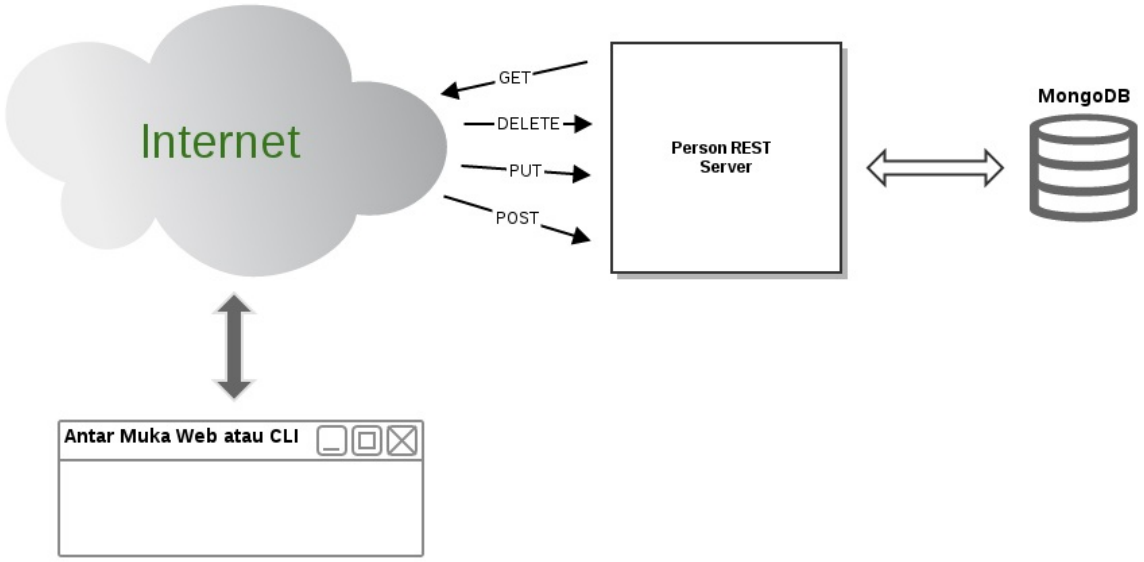
Method	URL	Action
GET	/persons	Get all persons data
GET	/persons/user_name	Get person data based on it's username
POST	/persons	Create new person data
PUT	/persons/user_name	Update person data based on it's username
DELETE	/persons/user_name	Delete person

Cara Kerja

Cara kerja dari aplikasi Person REST ini cukup mudah. Hanya saja untuk antar muka dengan pengguna tidak melalui antar muka web seperti halnya kita mengakses halaman Facebook misalnya, karena secara umum aplikasi REST fungsinya lebih ke memberikan layanan data mentah sehingga developer bertanggung jawab penuh untuk apa data-data tersebut digunakan, apakah akan ditampilkan ke browser web atau digunakan pada aplikasi mobile android atau akan digunakan untuk mengaktifkan device elektronik seperti Arduino, Raspberry Pi dll.

Untuk aplikasi Person REST ini, data disimpan di database MongoDB melalui operasi CRUD (*Create, Read, Update, Delete*) yang dapat diakses melalui *request* API yang telah di sebutkan sebelumnya.

Diagram kerja aplikasi Person REST digambarkan pada diagram dibawah ini



Pengetesan operasi CRUD untuk aplikasi bisa dengan menggunakan bantuan alat *Command Line Interface* seperti cURL atau HTTPie ataupun tool yang lebih ramah seperti Postman dan bisa juga anda membangun antar muka web dengan memakai API yang disediakan oleh server Person ini. Contoh pengetesan untuk aplikasi Person REST ini bisa lihat pada sub-bab **Pengetesan**.

Server

Kode sumber dari aplikasi ini dapat di download di link berikut

https://github.com/junwatu/rest-node-mongoose-mongodb

Instalasi

Clone kode sumber melalui git dan instal depedensi paket melalui npm

```
$ git clone https://github.com/junwatu/rest-node-mongoose-mongodb.git

$ cd rest-node-mongoose-mongodb

$ npm install
```

Pastikan database MongoDB sudah berjalan pada sistem anda jika menggunakan `service` **systemd** pada linux anda bisa menjalankannya dengan perintah berikut

```
$ sudo service mongod start
```

Kode

Untuk lingkungan produksi ada baiknya untuk memecah file kode ke bagian yang lebih kecil supaya lebih mudah dalam hal *maintenance*. Untuk kemudahan dan kesederhanaan aplikasi ini maka kode utama ditulis dalam satu file.

> app.js

```
/*
 * Koneksi Nodejs dengan MongoDB menggunakan Mongoose
 *
 * Author By Equan Pr.
 * http://equan.me
 *
 * License :  Whatever you want! :D
 */

var express = require("express"),
    app = express(),
    mongoose = require('mongoose'),
```

```
    path = require('path'),
    engines = require('consolidate');

app.configure(function () {
    app.use(express.logger());

    app.use(function(req, res, next) {
        res.header('Access-Control-Allow-Origin', '*');
        res.header('Access-Control-Allow-Methods', 'GET,PUT,POST,DELETE');
        res.header('Access-Control-Allow-Headers', 'Content-Type')
        if ('OPTIONS' == req.method) {
            res.send(200);
        }
        else {
            next();
        }
    })

    app.use(express.bodyParser());
    app.use(express.methodOverride());
    app.use(express.static(__dirname+'/public'));

    app.engine('html', engines.handlebars);

    app.set('views', __dirname + '/views');
    app.set('view engine', 'html');

    app.set('PORT', process.env.PORT || 5000);
    app.set('MONGODB_URI', process.env.MONGOLAB_URI ||
        process.env.MONGOHQ_URL || 'mongodb://localhost/persons');

});

/**
 * MongoDB connection
 */
var db = mongoose.createConnection(app.get('MONGODB_URI'));

db.on('connected', function () {
    console.log('Connected to MongoDB.');

});

db.on('error', function (err) {
    console.error.bind(console, 'Connection to MongoDB error!.');
});

db.on('close', function () {
    console.log('Connection to MongoDB closed.');
});

// Schema
var PersonsSchema = new mongoose.Schema({
```

```
        name: 'string',
        username: 'string',
        website: 'string',
        createdAt: 'date',
        updatedAt: 'date'
    }),

    Persons = db.model('Persons', PersonsSchema);

// Routes
app.get("/", function (req, res) {
    res.render('index',{
        data: 'Silly RESTful sample app built with Node.js, Express, Mongoose and Mong
oDB. ' +
            'Maybe it\'s useful for beginners ;)'
    });
});

// GET /persons
app.get("/persons", function (req, res) {
    // Find All
    Persons.find(function (err, persons) {
        if (err) res.json({error: err})

        if(persons)
            res.json({persons: persons});
    })
});

// POST /persons
app.post("/persons", function(req, res){
    /**
     * Get data from post
     * @type {Persons}
     */
    var person = new Persons({
        name: req.body.name,
        username: req.body.username,
        website: req.body.website,
        createdAt: new Date(),
        updatedAt: new Date()
    });

    person.save(function (err, person) {
        if (err) {
            res.send({error:err});
        }else {
            console.log('Save data: ' + person);
            res.json({message: ' save ok'});
        }
    })
});
```

```
// GET /persons/:username
app.get('/persons/:username', function(req, res){
    var param_username = req.params.username;

    Persons.find({username:param_username}, function(err, person){
        if(err) {
            res.json({
                data:"Error finding person."
            });
        }else {
            res.json({
                person: person
            });
        }
    })
});

// PUT /persons/:username
app.put('/persons/:username', function(req, res){
    var query = {username: req.params.username},
        data_update = {
            name : req.body.name,
            username: req.params.username,
            website: req.body.website,
            updatedAt: new Date()
        }

    Persons.update(query, data_update, {multi:false}, function(err, numberAffected, ra
wResponse ){
        if(err) {
            res.json({
                error:err
            })
        }else {
            res.json({
                numberAffected:numberAffected,
                rawResponse: rawResponse
            });
        }
    });

});

// DELETE /persons/:username
app.delete('/persons/:username', function(req, res){
    var param_username_del = req.params.username;

    Persons.remove({username:param_username_del}, function(err){
        if(err){ res.json({
            error:err
        })
        }else {
            res.json({message: "delete ok"});
```

```
        }
    });
});


app.listen(app.get('PORT'));
console.log("Server Port: " + app.get('PORT'));
```

Pengetesan

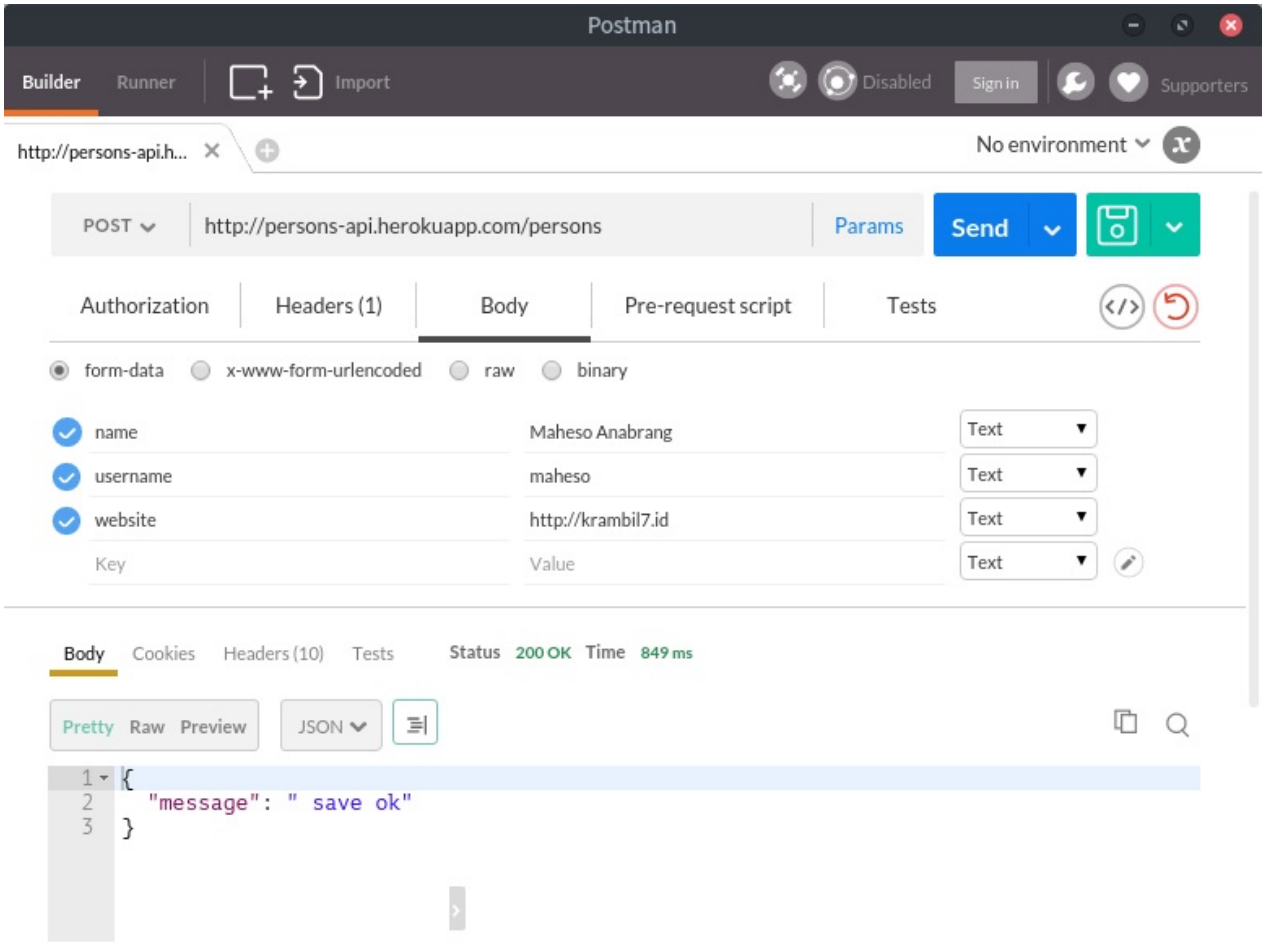
Untuk menggunakan atau pengetesan API ini cara termudah adalah dengan memakai Postman atau jika anda sudah terbiasa memakai tool terminal seperti cURL atau Httpie silahkan saja.

Demo aplikasi berada pada link berikut,

> persons-api.herokuapp.com

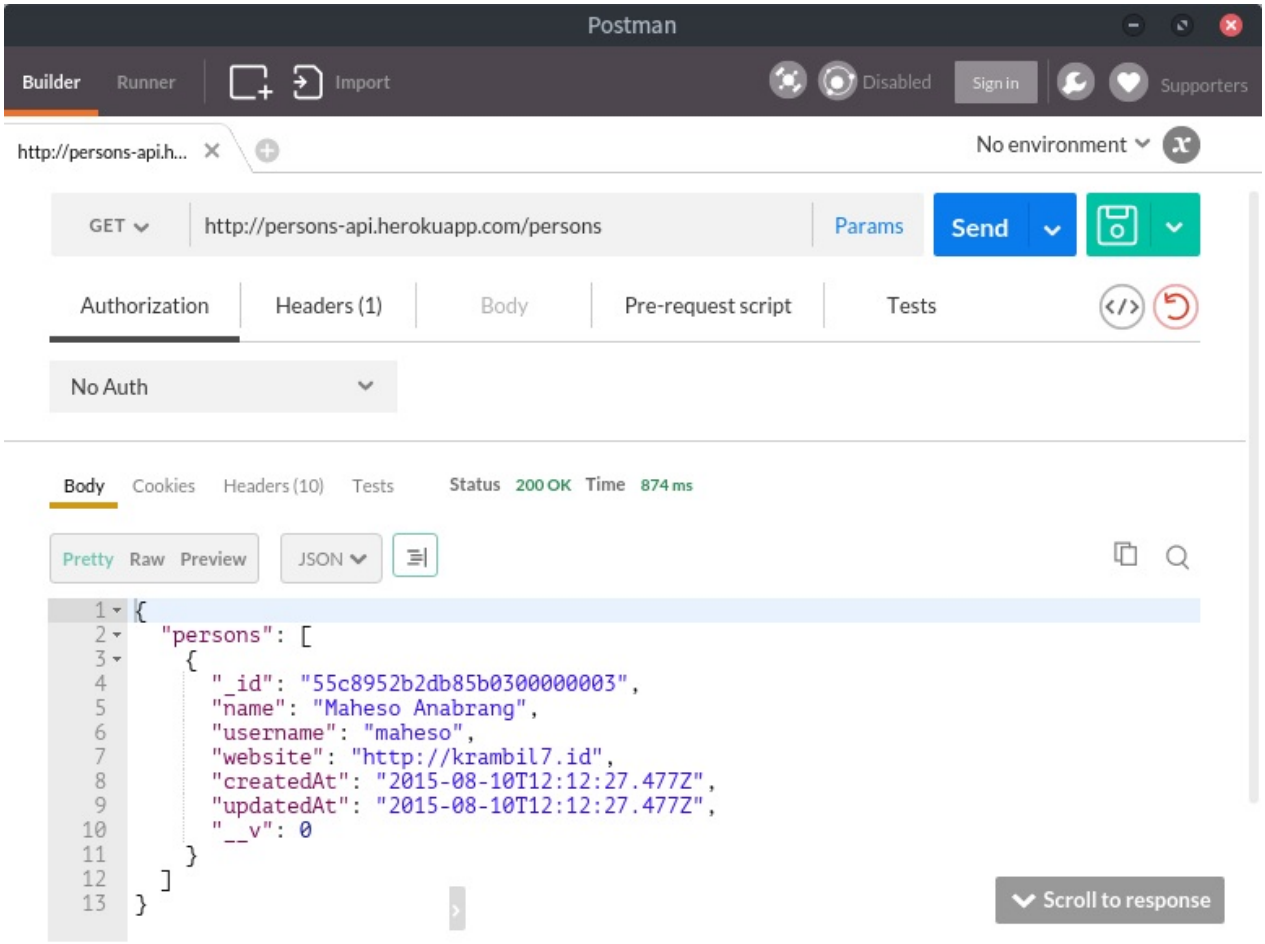
POST /persons

Untuk membuat data Person baru melalui api `/persons` cukup dengan memakai request POST.



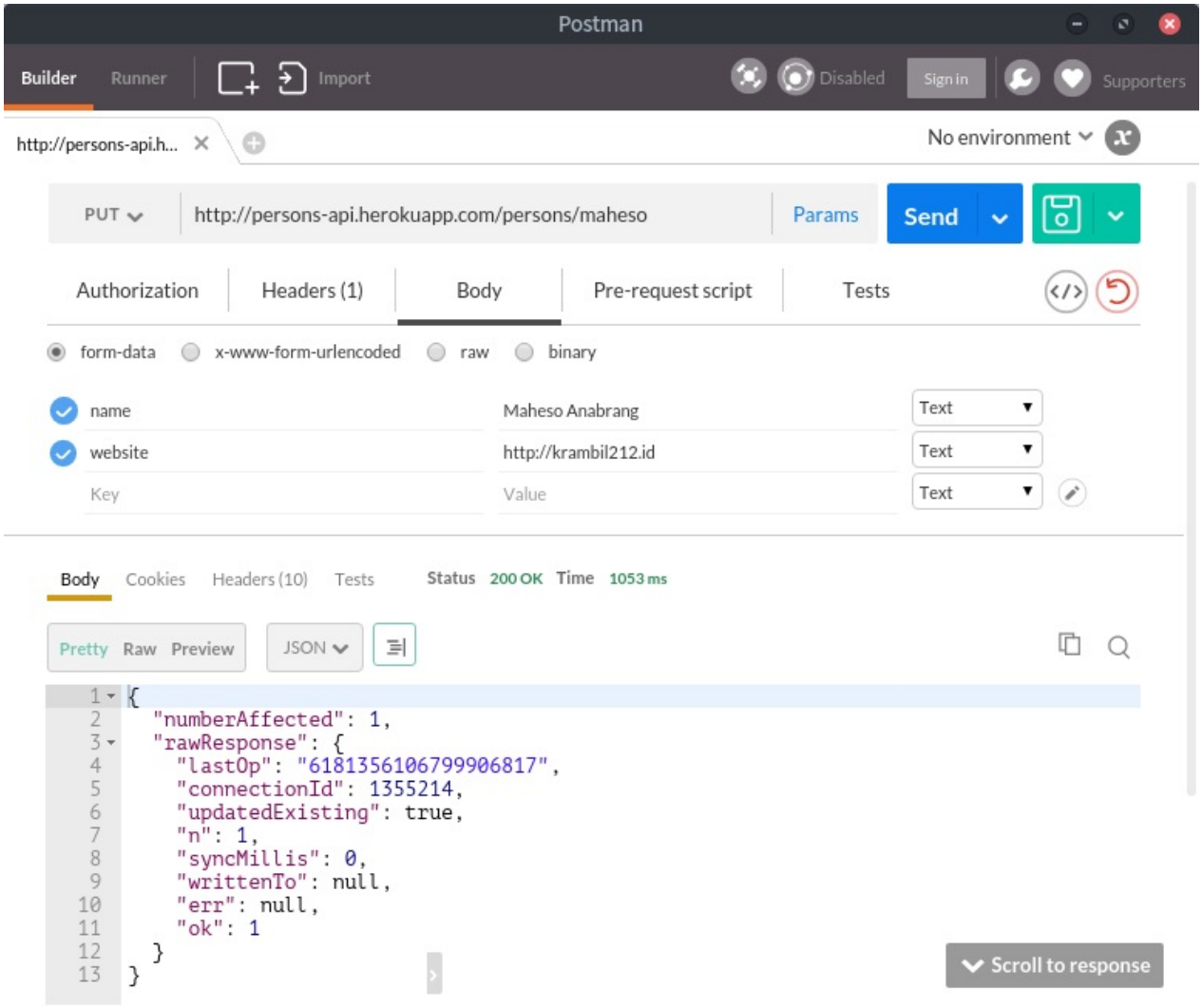
GET /persons

Mengambil data dengan mengakses API `/persons` yang akan mengembalikan semua data yang telah tersimpan sebelumnya.



PUT /persons/:username

Update data bisa dilakukan dengan mudah dengan memakai PUT.



DELETE /persons/:username

Operasi penghapusan data hanya bisa dilakukan satu persatu melalui key `:username`.

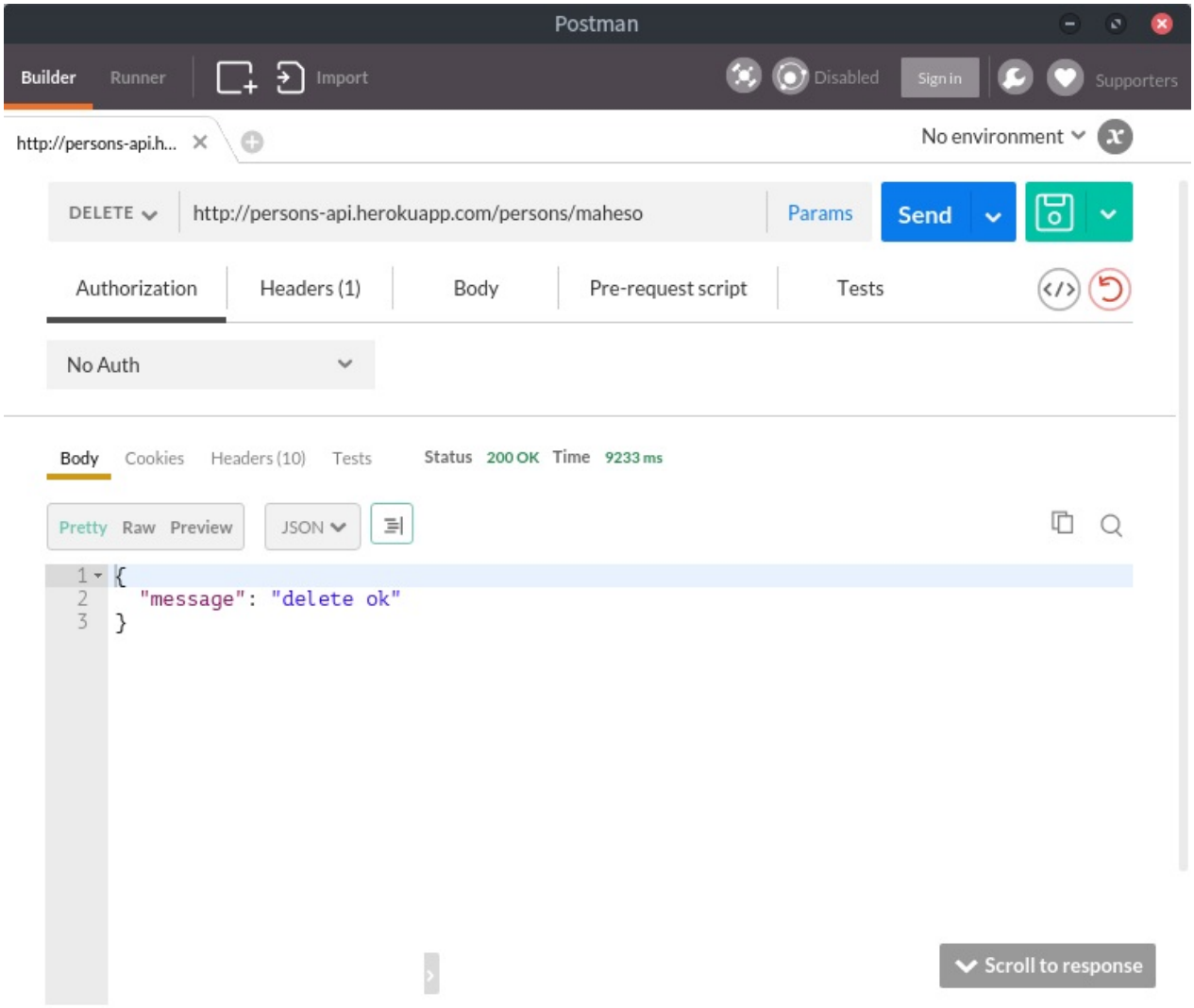
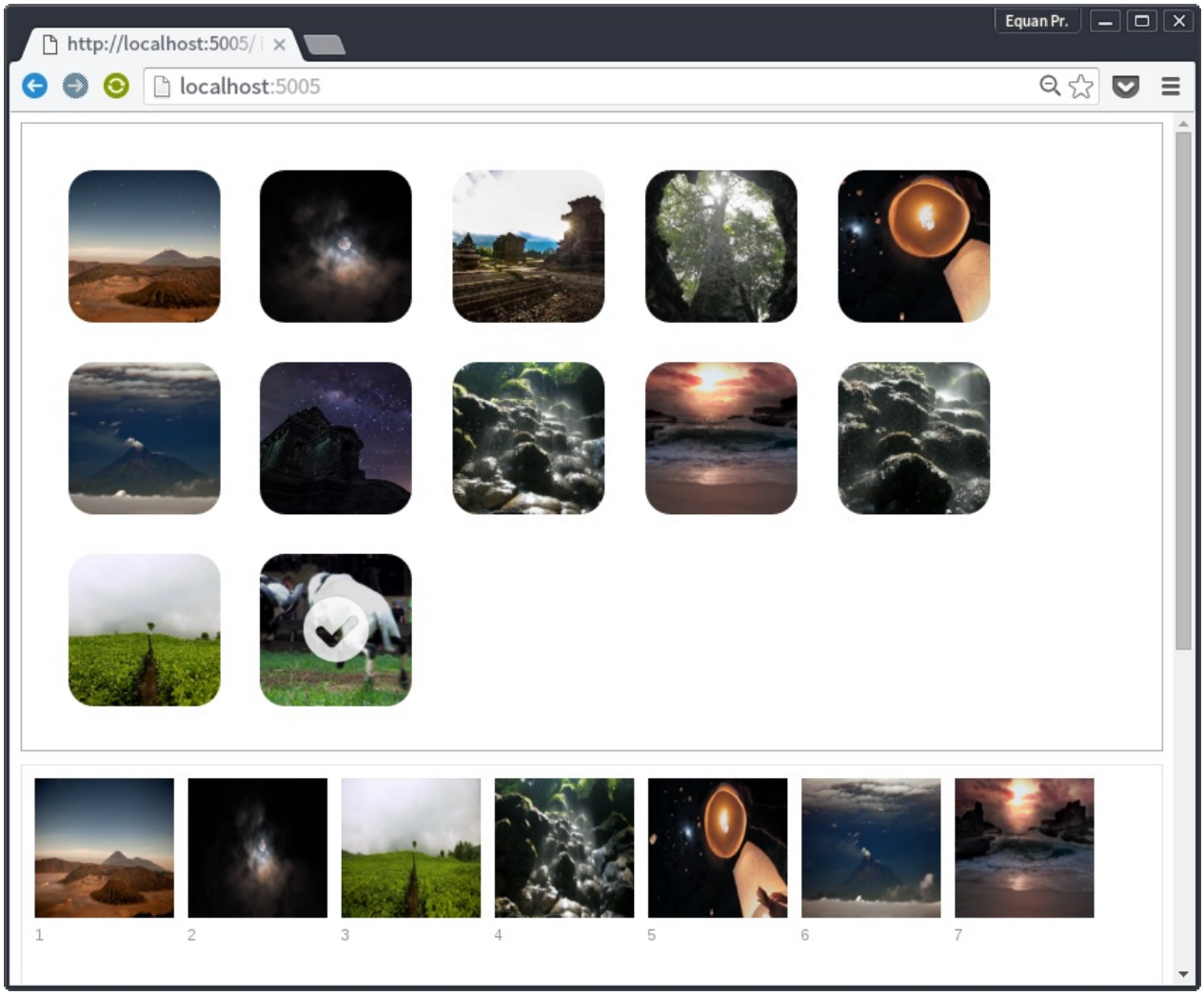

Postman
Builder
Runner
Import
Disabled
Sign in
Supporters
http://persons-api.h...
No environment
DELETE
http://persons-api.herokuapp.com/persons/maheso
Params
Send
Authorization
Headers (1)
Body
Pre-request script
Tests
No Auth
Body
Cookies
Headers (10)
Tests
Status 200 OK
Time 9233 ms
Pretty
Raw
Preview
JSON
1 {
2 "message": "delete ok"
3 }
Scroll to response

Image Uploader



Aplikasi ini sangat sederhana, cara kerjanya yaitu gambar di upload ke server dan kemudian ditampilkan kembali ke browser.

Server

Pada sisi server aplikasi ini memakai framework ExpressJS untuk menangani request HTTP dan paket formidable untuk menangani file yang diupload.

> Catatan: Aplikasi ini memakai sedikit fitur ES6 seperti `let`, `const`, *arrow function* sehingga anda perlu menginstall setidaknya Node.js v4.2.1 LTS

```
server.post('/upload', (req, res) => {
  let form = new formidable.IncomingForm()
  form.uploadDir = path.join(__dirname, 'uploads')
  form.hash = true
  form.multiples = false
  form.keepExtensions = true

  form.parse(req, (err, fields, files) => {
    if (!err) {
      console.log(files.file.name)
      console.log(files.file.path)
      console.log(files.file.type)
    }
    res.end()
  })
})
```

Kode diatas akan menangani file yang akan diupload dan menyimpan hasil upload pada direktori `uploads`. Formidable dapat dengan mudah dikonfigurasi, lihat Github Formidable.

Uploader

Pada sisi klien uploder dibangun dengan memakai pustaka DropzoneJS yang mendukung *drag n drop* dan *preview thumbnail*. Pustaka ini sangat mudah untuk digunakan dan di kustomisasi

```
Dropzone.options.mydropzone = {
  init: function () {
    this.on("complete", function(file) {
      updateImage()
    })
  },
  maxFileSize : 2,
  acceptedFiles: 'image/*'
}
```

Konfigurasi diatas mengakibatkan uploader hanya menerima file bertipe gambar dan ukuran file tidak lebih dari 2MB dan yang perlu dicatat yaitu ketika file selesai diupload yaitu dengan mendengarkan event `complete` maka *list view* gambar yang dibuat dengan Kendo UI harus diupdate dengan memanggil metode `updateImage()`.

Image List

Kendo akan mengambil data dari server kemudian secara otomatis akan mengupdate `#listView` sesuai dengan banyaknya gambar yang telah terupload.

```
var updateImage = function () {
  var dataSource = new kendo.data.DataSource({
    transport: {
      read: {
        url: document.location.href+'service/images',
        dataType: 'json'
      }
    },
    pageSize: 21
  })

  $('#pager').kendoPager({
    dataSource: dataSource
  })

  $('#listView').kendoListView({
    dataSource: dataSource,
    template: kendo.template($('#template').html())
  })
}
```

Komponen Kendo UI yang dipakai adalah `ListView` dan framework UI ini seperti framework kebanyakan lainnya juga memakai template untuk menghasilkan UI secara dinamik.

```
 <div class="demo-section k-content wide">
        <div id="listView"></div>
        <div id="pager" class="k-pager-wrap"></div>
    </div>

    <script type="text/x-kendo-template" id="template">
      <div class="product">
        <img src="/#=ImageName#"/>
        <h3>#:ImageId#</h3>
      </div>
    </script>
```

Data yang akan diambil dari server yaitu data gambar harus mempunyai field `ImageName` dan `ImageId`.

Komponen yang disediakan oleh Kendo cukup lengkap dan jika anda tertarik dengan Kendo UI lebih lanjut silahkan berkunjung ke website resmi Telerik.

Untuk mengambil daftar gambar yang telah diupload klien akan mengakses URL berikut

```
http://localhost:5005/service/images
```

Server hanya akan memfilter file dengan tipe `jpg` dan `png` pada folder `uploads`. Filter dilakukan dengan mengecek tipe file melalui paket `mime` tepatnya melalui metode `mime.lookup(image)`.

```
server.get('/service/images', (req, res) => {
  let images = []
  fs.readdir(upload_dir, (err, files) => {
    if (!err) {
      for (let imageIndex = 0; imageIndex < files.length; imageIndex++) {
        let ftype = mime.lookup(path.join(upload_dir, files[imageIndex]))
        if (filetypes.indexOf(ftype) !== -1) {
          let data = {}
          data.ImageId = imageIndex
          data.ImageName = files[imageIndex]
          images.push(data)
        }
      }
      res.json(images)
    } else {
      res.end()
    }
  })
})
```

Data yang dikembalikan ke klien adalah data JSON dengan format seperti berikut

```
[{
  ImageId: 1,
  ImageName: 'file.jpg'
}]
```

Kode sumber dari aplikasi ini bisa anda dapatkan pada repo Github Image Uploader.

ECMAScript 2015 atau ES6

Meskipun ECMAScript 2015 atau yang lebih dikenal ES6 sudah diresmikan tetapi implementasi di browser dan platform Node.js masih terjadi secara *gradual*. Pada saat buku ini ditulis pada platform Node.js 4.x (LTS) dan 5.x sudah banyak fitur-fitur ES6 yang *built-in* dan untuk fitur yang belum didukung anda bisa memakai tool yang bernama Babel.

Untuk mempelajari ES6 banyak sekali resource online yang menyediakan jadi silahkan anda bereksplorasi dan mencoba konsep-konsep baru yang ditawarkan oleh JavaScript. Anda bisa memulai dari buku online gratis tentang ES6

- ExploringJS ES6

Pada dasarnya Babel berfungsi untuk mengkompilasi ES6 (atau ES7, 8) menjadi ES5 sehingga bisa dijalankan pada browser atau platform Node.js yang belum mendukung ES6. Salah satu pertanyaan terbesar kalau anda memakai Babel adalah bagaimana mengkompilasi hanya beberapa fitur ES6 saja jikalau browser ataupun Node.js yang akan kita pakai secara native sudah mendukung sebagian atau beberapa fitur ES6?

Babel on The Fly

Sejak di release Babel 6, tool ini mendukung kompilasi ES6 berdasarkan plugin artinya anda bisa memilih fitur mana yang akan dikompilasi ke ES5 jika misalnya pada platform Node.js sudah mendukung banyak fitur ES6.

Kalau anda pernah memakai `babel-node` make sekarang diganti dengan paket `babel-cli`

```
$ npm install -g babel-cli
```

Apa kegunaannya? `babel-cli` sangat berguna untuk mengkompilasi ES6 secara `on the fly` misalnya begini

lib/module.js

```
'use strict'
export function test() {
  console.log('test')
}
```

main.js

```
'use strict'
import {test} from './lib/module'
test()
```

Maka untuk mengeksekusi file `main.js` bisa melalui `babel-node`

```
$ babel-node main.js
```

Jika anda memakai project dengan ES6 selain `babel-cli` maka paket berikut juga perlu di install

```
$ npm install --save-dev babel babel-core babel-loader
```

dan juga setting file `.babelrc` (meskipun sebenarnya ada beberapa alternatif lain dimana harus menuliskan setting babel tapi cara ini lebih UNIX, tergantung selera masing masing)

.babelrc

```
{
  "plugins": ["transform-es2015-modules-commonjs"]
}
```

Bisa anda lihat bahwa plugins yang akan kita gunakan adalah `transform-es2015-modules-commonjs` jadi harus di install juga

```
$ npm install --save-dev babel-plugin-transform-es2015-modules-commonjs
```

Lalu bagaimana kalau kita menginginkan semua plugin kita pakai? jawabannya adalah dengan memakai `presets`

```
$ npm install --save-dev babel-preset-es2015
```

dan `.babelrc` perlu anda rubah menjadi seperti dibawah ini

```
{
  "presets": ["es2015"]
}
```

Catatan Penjelasan diatas berlaku untuk JavaScript yang digunakan pada sisi server untuk client atau JavaScript yang berjalan pada browser anda bisa memakai webpack bersama dengan babel.

Tentang Pengarang

Equan Pr. adalah developer NodeJS dan peminum kopi kelas berat. Selalu sibuk dengan yang berbau JavaScript ketika di depan komputer, penggila film & musik metal dan kadang-kadang nge-twit di @junwatu atau nge-Github.