



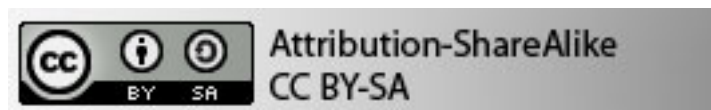
Pengembangan Aplikasi Cloud Computing Menggunakan Node.js

Oleh:

BAMBANG PURNOMOSIDI D. P. ([HTTP://BPDP.NAME](http://BPDP.NAME))

Kontributor:

AJI KISWORO MUKTI ([HTTP://ADZYMANIAC.WEB.ID](http://ADZYMANIAC.WEB.ID))



Buku bebas dengan lisensi Creative Common BY-SA. Silahkan membaca isi
selengkapnya serta penjelasannya di
[http://wiki.creativecommons.org/Licenses/by-sa/3.0LegalText_\(Indonesian\)](http://wiki.creativecommons.org/Licenses/by-sa/3.0LegalText_(Indonesian))

May 6, 2013

Kata Pengantar

Buku bebas ini merupakan buku yang dirancang untuk keperluan memberikan pengetahuan mendasar pengembangan aplikasi berbasis Cloud Computing, khususnya menggunakan Node.js. Pada buku ini akan dibahas penggunaan Node.js untuk mengembangkan aplikasi SaaS (*Software as a Service*). Node.js merupakan software di sisi server yang dikembangkan dari *engine* JavaScript V8 dari Google serta **libuv** (<https://github.com/joyent/libuv>)¹.

Jika selama ini kebanyakan orang mengenal JavaScript hanya di sisi klien (browser), dengan Node.js ini, pemrogram bisa menggunakan JavaScript di sisi server. Meskipun ini bukan hal baru, tetapi paradigma pemrograman yang dibawa oleh Node.js dengan *evented - asynchronous I/O* menarik dalam pengembangan aplikasi Web (selain kita hanya perlu menggunakan 1 bahasa yang sama di sisi server maupun di sisi klien).

Untuk mengikuti materi yang ada pada buku ini, pembaca diharapkan menyiapkan peranti komputer dengan beberapa software berikut terpasang:

- ✓ Sistem operasi Linux (distribusi apa saja) - lihat di <http://www.distrowatch.com>. Sistem operasi Linux ini bukan keharusan, anda bisa menggunakan Windows tetapi silahkan membuat penyesuaian-penyesuaian sendiri yang diperlukan. Kirim saya *pull request* jika anda menuliskan pengalaman anda di Windows!
- ✓ Git (untuk *version control system*) - bisa diperoleh di <http://git-scm.com>. Saya menggunakan versi 1.8.2.2.
- ✓ Ruby (<http://www.ruby-lang.org/en/>) - diperlukan untuk menginstall dan mengeksekusi *vmc*, perintah *command line* untuk mengelola aplikasi Cloud di CloudFoundry. Versi Ruby yang digunakan adalah versi 2.0.0p0 (2013-02-24 revision 39474).
- ✓ mongoDB (basis data NOSQL) - bisa diperoleh di <http://www.mongodb.org>, saya menggunakan versi 2.4.4-pre-
- ✓ Vim (untuk mengedit source code) - bisa diperoleh di <http://www.vim.org>. Jika tidak terbiasa menggunakan Vim, bisa menggunakan editor teks lainnya (atau IDE), misalnya gedit (ada di GNOME), geany (<http://geany.org>), KATE (ada di KDE), dan lain-lain.

Software utama untuk keperluan workshop ini yaitu Node.js serta command line tools dari provider Cloud Computing (materi ini menggunakan fasilitas dari CloudFoundry), akan dibahas pada bab-bab tertentu. Materi akan lebih banyak berorientasi ke command line / shell sehingga para pembaca sebaiknya sudah memahami cara-cara menggunakan shell di Linux. Anda bisa menggunakan shell apa saja (bash, tcsh, zsh, ksh, dan lain-lain), saya menggunakan bash 4.2.45(2)-release.

Have fun!

¹Versi sebelum 0.9.0 menggunakan **libev** dari Mark Lechmann

Kata Pengantar	i
Daftar Gambar	iii
Listing Program	iv
Daftar Tabel	v
1. Pengenalan Cloud Computing dan Infrastruktur Pengembangan Aplikasi Berbasis Node.js	1
1.1. Apa itu Cloud Computing?	1
1.2. Karakteristik Cloud Computing	1
1.3. <i>Public</i> dan <i>Private</i> Cloud Computing	3
1.4. Model Layanan Cloud Computing	3
1.5. Pengembangan Aplikasi di Cloud Computing	3
1.6. Node.js dan Cloud Computing	4
1.7. Layanan Hosting Aplikasi: CloudFoundry	4
1.7.1. Pendaftaran	4
1.7.2. Instalasi <i>Command Line Utilities</i>	4
1.7.3. Konfigurasi di Server Cloud	10
1.7.4. Instalasi dan Konfigurasi Node.js di Komputer Lokal	12
1.8. Pengelolaan Aplikasi di Cloud	13
1.8.1. <i>Push, Delete, Update</i> Aplikasi	14
1.8.2. Menggunakan Framework ExpressJS	14
1.8.3. Tanpa Framework	19
2. REPL dan Dasar-dasar JavaScript di Node.js	21
2.1. REPL	21
2.1.1. Mengaktifkan REPL	21

2.1.2.	Perintah-perintah REPL	22
2.2.	Dasar-dasar JavaScript di Node.js	23
2.2.1.	Membaca <i>Masukan</i> dari Stream / <i>Masukan Standar (stdin)</i>	23
2.2.2.	Nilai/Value dan Tipe Data	24
2.2.3.	Variabel	25
2.2.4.	Konstanta	25
2.2.5.	Fungsi	26
	Pengertian Fungsi	26
	Definisi Fungsi	26
	Fungsi Anonim	27
	Fungsi Rekursif	27
	Fungsi di dalam Fungsi / <i>Nested Functions</i>	27
2.2.6.	Literal	28
	Literal Array	28
	Literal Boolean	29
	Literal Integer	29
	Literal Floating-point	29
	Literal Obyek	29
	Literal String	29
2.2.7.	Struktur Data dan Representasi JSON	30
2.2.8.	Aliran Kendali	31
	Pernyataan Kondisi <i>if .. else if .. else</i>	31
	Pernyataan <i>switch</i>	32
	<i>Looping</i>	33
2.2.9.	Penanganan Error	35
3.	Paradigma Pemrograman di JavaScript	37
3.1.	Pemrograman Fungsional	37
3.1.1.	Ekspresi Lambda	37
3.1.2.	Higher-order Function	38
3.1.3.	Closure	39
3.1.4.	Currying	39
3.2.	Pemrograman Berorientasi Obyek	40
3.2.1.	Pengertian	40
3.2.2.	Definisi Obyek	40
3.2.3.	<i>Inheritance</i> / Pewarisan	41
4.	Mengelola Paket Menggunakan npm	43
4.1.	Apakah npm Itu?	43
4.2.	Menggunakan npm	43
4.2.1.	Instalasi Paket	44
4.2.2.	Struktur Instalasi Paket Node.js	44
4.2.3.	Menghapus Paket / <i>Uninstall</i>	45
4.2.4.	Mencari Paket	45

4.2.5. Menampilkan Informasi Paket	46
4.2.6. Memperbaharui Paket	47
5. Node.js dan Web: Teknik Pengembangan Aplikasi	48
5.1. Pendahuluan	48
5.2. <i>Event-Driven Programming</i> dan <i>EventEmitter</i>	49
5.3. Asynchronous / Non-blocking IO dan <i>Callback</i>	50
6. Mengakses Basis Data NoSQL: mongoDB	52
6.1. Apa itu Basis Data NoSQL?	52
6.2. Mengenal mongoDB dan Fitur-fiturnya	52
6.2.1. Memulai Server	53
6.2.2. Klien dan Shell mongoDB	54
6.2.3. Documents dan Collections	56
6.3. Node.js dan MongoDB	57
6.3.1. Node-gyp	57
6.3.2. Driver Node.js untuk mongoDB	57
6.3.3. Mengakses mongoDB dari Node.js	58
6.4. Aplikasi Web Menggunakan Node.js dan mongoDB	59
7. Pola Arsitektur Aplikasi Web: MVC dan ExpressJS	62
7.1. Apa itu Pola Arsitektur?	62
7.2. Pola Arsitektur MVC	62
7.3. Implementasi Pola Arsitektur MVC Menggunakan ExpressJS	64
7.3.1. Struktur Aplikasi	64
7.3.2. File-file yang Diperlukan	65
7.3.3. Hasil	66
7.4. Pola Arsitektur Aplikasi Web Lain dan Implementasinya	66
8. Real-time Web Menggunakan Socket.io	68
8.1. Apa itu Real-time Web?	68
8.2. Teknologi Pendukung Real-time Web	68
8.2.1. <i>Ajax Technology</i>	68
8.2.2. Comet dan <i>Push Technology</i>	69
SSE (<i>Server-Sent Events</i>)	69
Bayeux Protocol	69
BOSH Protocol	69
8.2.3. WebSocket	70
8.3. Socket.io	70
8.3.1. Apa itu Socket.io?	70
8.3.2. Menggunakan Socket.io untuk Real-time Web	71
Tentang Aplikasi	71
Membuat Kerangka Aplikasi dengan ExpressJS	71
Instalasi Paket yang Diperlukan	71

Konfigurasi JavaScript untuk Browser	72
Hapus File yang Tidak Diperlukan	72
Ubah File-file Tertentu	72
Menjalankan Server Socket.io	73
Daftar Pustaka	76
Lampiran	78
A. Gaya Penulisan Kode / <i>Coding Style</i>	78
A.1. Tentang Gaya Penulisan Kode	78
A.2. npm's Coding Style	78
A.2.1. DESCRIPTION	78
A.2.2. Line Length	79
A.2.3. Indentation	79
A.2.4. Curly braces	79
A.2.5. Semicolons	79
A.2.6. Comma First	80
A.2.7. Whitespace	80
A.2.8. Functions	80
A.2.9. Callbacks, Sync/async Style	81
A.2.10. Errors	81
A.2.11. Logging	81
A.2.12. Case, naming, etc.	81
A.2.13. null, undefined, false, 0	81
B. <i>Commit History</i> dari Penulis Utama dan Kontributor	83
Indeks	85

1.1. Model Cloud Computing	2
1.2. Pendaftaran di CF	5
1.3. Hasil proses pendaftaran di CF	6
1.4. E-mail persetujuan dan pemberitahuan <i>credentials</i>	7
1.5. Hasil push ke server	17
1.6. Hasil update dengan menyertakan versi Node.js	18
1.7. Deployment menggunakan versi runtime tertentu	19
1.8. Hasil deployment app.js tanpa framework	20
4.1. Tampilan “npm ls” pada direktori proyek dengan paket terinstall lengkap	46
6.1. Admin web console untuk mongoDB	55
7.1. Pola arsitektur MVC	63
7.2. Pola arsitektur MVC	67
8.1. Hasil di browser dari ExpressJS + Socket.io	75

Listing Program

1.1. Instalasi vmc	5
1.2. Peringatan setting PATH untuk vmc	8
1.3. Peringatan setting PATH untuk vmc	8
1.4. Hasil gem yang terinstall	8
1.5. Hasil opsi help dari vmc	9
1.6. Mengubah target server - belum ada konfigurasi	10
1.7. Mengubah target server - setelah konfigurasi	10
1.8. Login ke server	10
1.9. Mengubah password server	12
1.10. Hasil dari download Node.js	12
1.11. Ekstraksi Node.js	13
1.12. Konfigurasi variabel lingkungan Node.js	13
1.13. Instalasi ExpressJS menggunakan npm	14
1.14. Perintah express	14
1.15. Menggunakan express untuk membuat rerangka aplikasi	14
1.16. package.json untuk ExpressJS	15
1.17. app.js untuk ExpressJS	15
1.18. Hasil edit routes/index.js	16
1.19. Deployment aplikasi ExpressJS ke CF	16
1.20. Update: menambahkan versi Node.js ke routes/index.js	17
1.21. Mengupdate aplikasi di server	17
1.22. Menghapus aplikasi yang di-deploy di CF	17
1.23. Deployment ke CF dengan memilih runtime Node.js	18
1.24. app.js tanpa framework	19
1.25. Deployment app.js tanpa framework	19
2.1. Node.js REPL	21
2.2. Contoh penggunaan .load dalam REPL	22

2.3. Contoh penggunaan perintah <code>.save</code> di sesi REPL	23
2.4. <code>readline.js</code> : penggunaan pustaka <code>Readline</code> untuk masukan	24
2.5. Fitur <i>dynamically typed language</i>	25
2.6. Contoh konstanta dalam JavaScript	25
2.7. Sintaksis Fungsi dalam JavaScript	26
2.8. Pemanggilan Fungsi dalam JavaScript	26
2.9. Contoh deklarasi fungsi dan pemanggilannya	26
2.10. Fungsi anonim	27
2.11. Fungsi rekursif untuk menghitung faktorial	27
2.12. Fungsi di dalam Fungsi	28
2.13. Array di JavaScript	28
2.14. Representasi <i>JSON</i>	30
2.15. Pernyataan <i>if .. else if .. else</i>	31
2.16. Pernyataan <i>Contoh penggunaan “switch”</i>	32
2.17. Pernyataan <i>for</i>	33
2.18. Pernyataan <i>for .. in</i>	33
2.19. Pernyataan <i>do .. while</i>	34
2.20. Pernyataan <i>while</i>	34
2.21. Pernyataan <i>break</i> dan <i>continue</i>	35
2.22. Pernyataan <i>break</i> dengan label	35
2.23. Pernyataan <i>try catch finally</i>	36
2.24. Pernyataan <i>try catch throw</i>	36
3.1. Ekspresi Lambda di JavaScript	38
3.2. Higher-order Function di JavaScript	38
3.3. Closure di JavaScript	39
3.4. Currying di JavaScript	39
3.5. Definisi obyek di JavaScript	40
3.6. Pewarisan di PBO JavaScript	41
4.1. Sintaksis lengkap perintah <i>npm</i>	43
4.2. Cara install paket menggunakan <i>npm</i>	44
4.3. Argumen <i>npm</i> untuk melihat daftar paket terpasang	45
4.4. <i>npm ls</i> pada aplikasi yang paket-paketnya belum terinstall	45
4.5. Perintah menghapus paket di <i>npm</i>	45
4.6. Perintah menghapus paket di <i>npm</i>	46
4.7. Menampilkan rincian suatu paket dalam format <i>JSON</i>	46
4.8. Memperbaharui paket	47
5.1. <code>pegawai.json</code>	48
5.2. <code>server.js</code>	49
5.3. <code>server-on-error.js</code>	50
5.4. Membaca file secara <i>synchronous</i>	50
5.5. Membaca file secara <i>asynchronous</i>	51

6.1. Menjalankan server MongoDB (mongod)	53
6.2. Mengakhiri server MongoDB (mongod)	54
6.3. Shell mongoDB (mongo)	54
6.4. Sesi dalam shell mongoDB	56
6.5. Instalasi node-gyp	57
6.6. Instalasi driver mongoDB	57
6.7. Instalasi wrapper mongojs	58
6.8. Mengakses mongoDB dari Node.js	58
6.9. Express+MongoDB web: app.js	59
6.10. Express+MongoDB web: package.json	60
6.11. Express+MongoDB web: routes/index.js	60
6.12. Express+MongoDB web: routes/employee.js	60
6.13. Express+MongoDB web: views/employee.jade	61
6.14. views/index.jade	61
7.1. Struktur direktori asli aplikasi ExpressJS	64
7.2. Struktur direktori ExpressJS sesuai pola MVC	64
7.3. app.js	65
7.4. controllers/user.js	65
7.5. models/db.js	66
7.6. package.json	66
8.1. package.json	71
8.2. Isi direktori dist di socket.io-client	72
8.3. app.js	72
8.4. views/index.jade	73
8.5. routes/index.js	73
8.6. Menjalankan server Socket.io	73
8.7. Kode sumber di browser	74
A.1. Bad curly braces placement - 1	79
A.2. Good curly braces placement - 1	79
A.3. Bad curly braces placement - 2	79
A.4. Good curly braces placement - 2	79
A.5. Good semicolon usage	80
A.6. Comma first	80
B.1. <i>Commit history</i>	83

Daftar Tabel

Pengenalan Cloud Computing dan Infrastruktur Pengembangan Aplikasi Berbasis Node.js

1.1. Apa itu Cloud Computing?

Cloud Computing, atau sering diterjemahkan sebagai “Komputasi Awan” dalam bahasa Indonesia mempunyai berbagai definisi:

- ✓ **Wikipedia**: penggunaan sumber daya komputasi (peranti keras dan peranti lunak) yang berfungsi untuk memberikan layanan melalui suatu jaringan (pada umumnya Internet)¹.
- ✓ **NIST**²: model yang memungkinkan akses jaringan ubiquitous (dari mana saja), nyaman, on-demand (saat ada permintaan) ke sekumpulan sumber daya komputasi yang dikonfigurasi untuk berbagi (jaringan, server, penyimpanan, dan berbagai layanan lain) yang dapat dengan cepat ditetapkan dan dirilis dengan usaha yang minimal dari manajemen ataupun interaksi dengan penyedia layanan³.

Jika diwujudkan secara visual, Cloud Computing bisa dilihat pada Gambar 1.1⁴

1.2. Karakteristik Cloud Computing

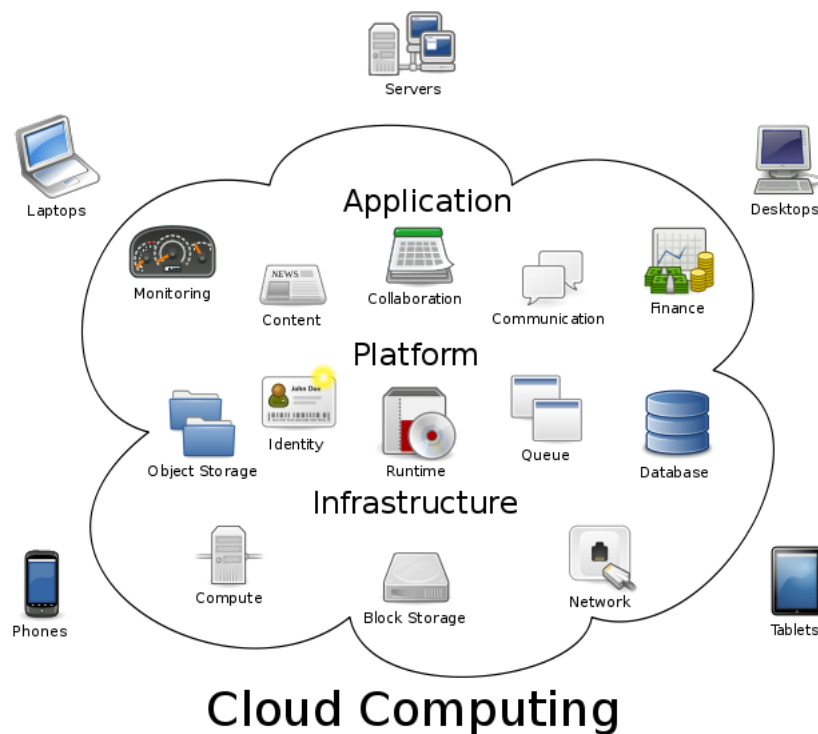
Menurut NIST, ada beberapa karakteristik dari Cloud Computing:

¹http://en.wikipedia.org/wiki/Cloud_computing

²The National Institute of Standards and Technology

³<http://csrc.nist.gov/publications/PubsSPs.html#800-145>

⁴Gambar dibuat oleh Sam Johnston, diambil dari http://en.wikipedia.org/w/index.php?title=File:Cloud_computing.svg&page=1



Gambar 1.1.: Model Cloud Computing

- ✓ **On-demand self-service:** layanan bisa diperoleh pada saat diminta, tanpa intervensi atau interaksi manusia di sisi penyedia jasa.
- ✓ **Broad network access:** tersedia melalui jaringan dengan berbagai peranti yang umum (komputer, tablet, HP, dan lain-lain)
- ✓ **Resource pooling:** sumber daya komputasi dari penyedia jasa terkumpul untuk melayani.
- ✓ **Rapid elasticity:** skalabilitas.
- ✓ **Measured service:** penggunaan sumber daya bisa diukur, di-monitor, dikendalikan, dan dilaporkan.

Karakteristik lain yang tidak kalah penting adalah *multitenancy*. *Multitenancy* merupakan suatu prinsip dalam arsitektur software. Pada arsitektur tersebut, satu instan dari software berjalan pada server, melayani banyak organisasi klien. Aplikasi dirancang untuk mempartisi data dan konfigurasinya secara virtual dan setiap organisasi klien tersebut

bekerja dengan instan aplikasi virtual tersebut⁵.

1.3. *Public dan Private Cloud Computing*

Cloud Computing bisa dibangun untuk keperluan pribadi suatu organisasi dan (secara legal) hanya bisa diakses oleh organisasi yang bersangkutan. Tipe tersebut dikenal dengan *Private Cloud Computing*. Sementara itu, jika sumber daya Cloud Computing bisa diakses oleh publik (dengan hak akses yang sesuai), maka model tersebut dikenal sebagai *Public Cloud Computing*. Pembahasan di buku ini adalah pembahasan tentang *Public Cloud Computing* dan semua referensi tentang Cloud Computing di buku ini akan menunjuk pada *Public Cloud Computing* kecuali dinyatakan lain.

1.4. Model Layanan Cloud Computing

Model layanan pada Cloud Computing akan berkembang sesuai kebutuhan konsumen serta inovasi dari berbagai penyedia layanan. Saat ini, pada umumnya, ada tiga model layanan:

- ✓ **SaaS** (*Software as a Service*): layanan berupa aplikasi yang ditempatkan pada infrastruktur penyedia layanan, siap digunakan oleh konsumen.
- ✓ **PaaS** (*Platform as a Service*): menyediakan layanan ke konsumen berupa platform untuk men-deploy aplikasi.
- ✓ **IaaS** (*Infrastructure as a Service*): menyediakan layanan ke konsumen berupa berbagai sumber daya komputasi (pemrosesan, penyimpanan, jaringan, dan sumber daya fundamental lainnya).

Meski sampai saat ini, umumnya terdapat tiga model tersebut, beberapa model kelihatannya sudah mulai muncul, misalnya STaaS (*Storage as a Service*), SECaaS (*Security as a Service*), DaaS (*Data as a Service*), TEaaS (*Test Environment as a Service*), *Desktop Virtualization*, APIaaS (*API as a Service*).

1.5. Pengembangan Aplikasi di Cloud Computing

Pada umumnya, para pengembang aplikasi di Cloud Computing juga menggunakan pendekatan *Agile Software Development* yang berbasis pada pengembangan secara iteratif untuk setiap *milestone* (dalam iterasi analisis-desain-coding-testing-debugging) mulai dari *milestone* paling awal sampai software dirilis. Perbedaan paling mendasar hanyalah pada platform yang digunakan untuk *deployment*, peranti pengembangan yang digunakan, serta utilitas untuk mengelola aplikasi yang di-deploy pada instan di cloud.

Pengembangan aplikasi di Cloud Computing akan melibatkan peranti pengembang yang didukung oleh infrastruktur Cloud. Kita akan memerlukan PaaS untuk keperluan ini. Pada dasarnya pengembangan aplikasi akan meliputi siklus berikut:

⁵<http://en.wikipedia.org/wiki/Multitenancy>

- ✓ *Coding*
- ✓ Test di komputer lokal
- ✓ Upload ke server (dalam Cloud Computing, proses ini diistilahkan dengan “*push*”)
- ✓ Edit - push

Jika pengembangan aplikasi dilakukan oleh tim, maka perlu adanya software untuk *version control*, misalnya Git, mercurial, dan lain-lain. Setelah itu, aktivitas yang dilakukan biasanya terpusat pada *push* (untuk mengupload instan dari aplikasi ke server) dan *pull* (untuk mengambil instan aplikasi dari server).

1.6. Node.js dan Cloud Computing

Node.js merupakan salah satu peranti pengembang yang bisa digunakan untuk membuat aplikasi berbasis Cloud. Node.js dikembangkan dari *engine* JavaScript yang dibuat oleh Google untuk browser *Chrome / Chromium* (V8) ditambah dengan libUV serta beberapa pustaka internal lainnya. Dengan menggunakan Node.js, semua pengembangan akan dilakukan menggunakan JavaScript, baik pada sisi klien maupun server. Node.js dibuat pertama kali oleh Ryan Dahl (twitter.com/ryah) dan sampai saat ini dikembangkan oleh komunitas sebagai software bebas dengan pendanaan utama dari Joyent, perusahaan tempat Ryan Dahl bekerja.

1.7. Layanan Hosting Aplikasi: CloudFoundry

Saat ini, mulai banyak penyedia layanan Cloud yang mendukung Node.js, diantaranya adalah CloudFoundry (<http://www.cloudfoundry.com>, selanjutnya akan kita sebut dengan CF). Buku ini akan menggunakan fasilitas dari CF. Daftar lengkap dari penyedia infrastruktur Node.js bisa dilihat pada <https://github.com/joyent/node/wiki/Node-Hosting>.

1.7.1. Pendaftaran

Untuk menggunakan fasilitas dari CF, kita akan mendaftar lebih dahulu di URL <https://my.cloudfoundry.com/signup> seperti yang terlihat pada Gambar 1.2.

Setelah itu, CF akan mengirimkan pemberitahuan bahwa proses pendaftaran selesai seperti di Gambar 1.3. *Credentials* atau informasi tentang akun kita di CF akan dikirimkan ke e-mail kita seperti pada Gambar 1.4.

1.7.2. Instalasi *Command Line Utilities*

Command Line Utilities / CLU adalah software yang dijalankan melalui shell / *command line / command prompt*. CLU untuk CF ini dibuat dengan menggunakan Ruby dan didistribusikan dalam bentuk *gem* sehingga untuk instalasi ini diperlukan ruby dan rubygem. Berikut adalah perintah untuk instalasi vmc (CLU dari CF).

Wit
cal
ma
ap
ch
an

Be
giv
fra
de
Sc
vFi
yor
Clc
ho
var

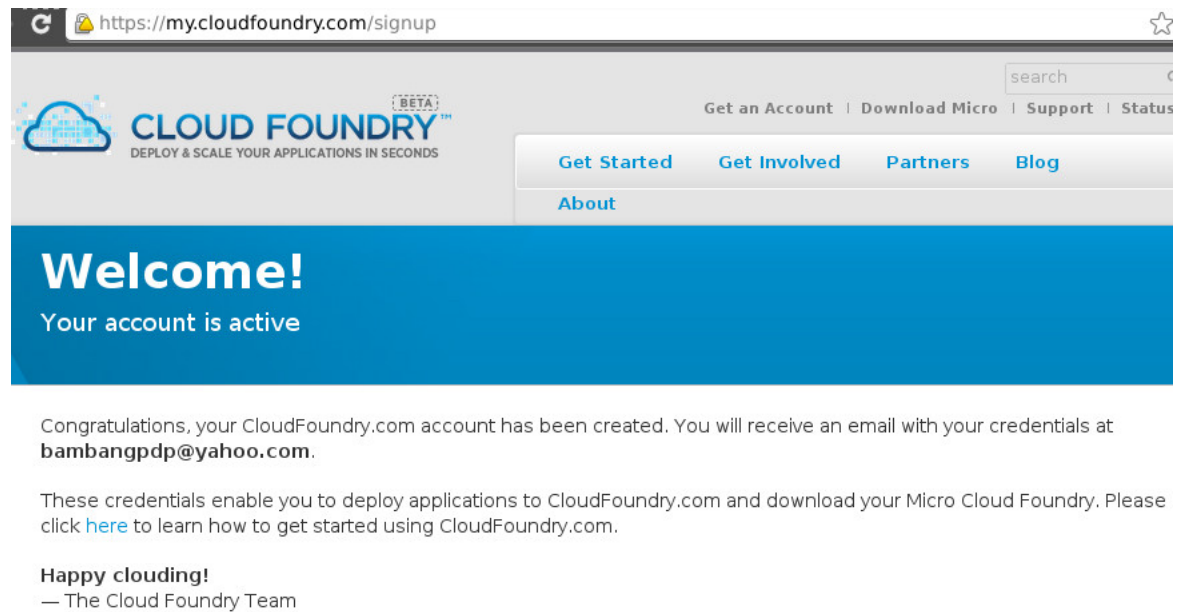
Wh
us
No

Gambar 1.2.: Pendaftaran di CF

```

1 $ gem search vmc --remote | grep "\bvmc\s"
2 vmc (0.5.0)
3 $ gem install vmc
4 Fetching: json_pure-1.7.7.gem (100%)
5 WARNING: You don't have /home/bdp/.gem/ruby/2.0.0/bin in your PATH,
6     gem executables will not run.
7 Successfully installed json_pure-1.7.7
8 Fetching: multi_json-1.7.2.gem (100%)
9 Successfully installed multi_json-1.7.2
10 Fetching: interact-0.5.2.gem (100%)
11 Successfully installed interact-0.5.2
12 Fetching: multipart-post-1.2.0.gem (100%)
13 Successfully installed multipart-post-1.2.0
14 Fetching: rubyzip-0.9.9.gem (100%)
15 Successfully installed rubyzip-0.9.9
16 Fetching: cf-uaa-lib-1.3.10.gem (100%)
17 Successfully installed cf-uaa-lib-1.3.10
18 Fetching: cfoundry-0.5.2.gem (100%)
19 Successfully installed cfoundry-0.5.2
20 Fetching: clouseau-0.0.2.gem (100%)
21 Successfully installed clouseau-0.0.2
22 Fetching: mothership-0.5.1.gem (100%)
23 Successfully installed mothership-0.5.1
24 Fetching: manifests-vmc-plugin-0.6.2.gem (100%)
25 Successfully installed manifests-vmc-plugin-0.6.2
26 Fetching: addressable-2.3.4.gem (100%)
27 Successfully installed addressable-2.3.4

```

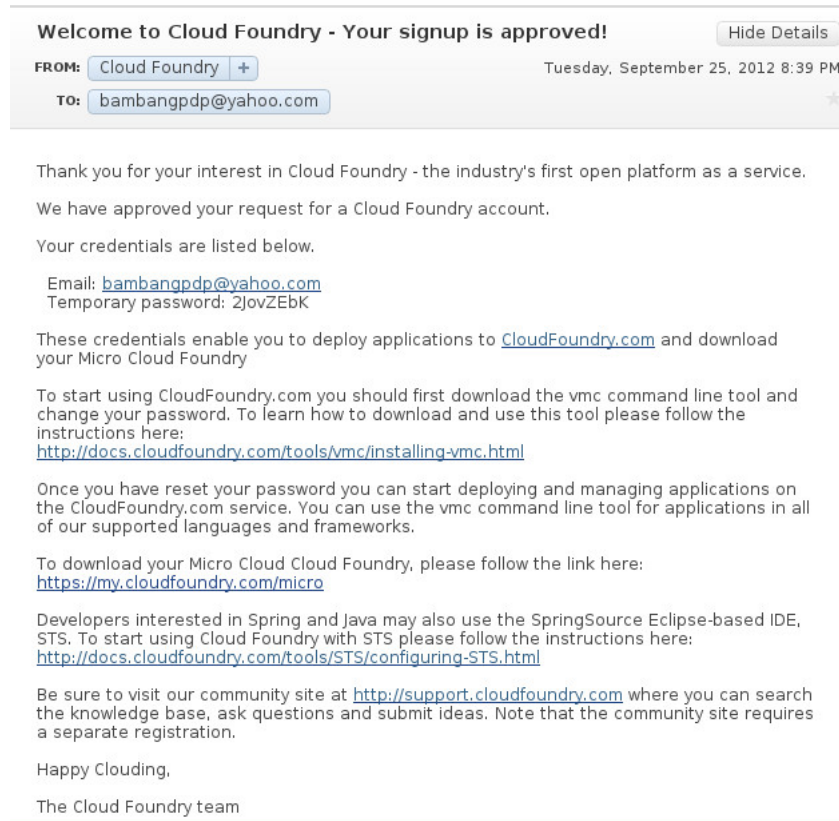



Gambar 1.3.: Hasil proses pendaftaran di CF

```

28 Fetching: caldecott-client-0.0.2.gem (100%)
29 Successfully installed caldecott-client-0.0.2
30 Fetching: mime-types-1.23.gem (100%)
31 Successfully installed mime-types-1.23
32 Fetching: rest-client-1.6.7.gem (100%)
33 Successfully installed rest-client-1.6.7
34 Fetching: uuidtools-2.1.4.gem (100%)
35 Successfully installed uuidtools-2.1.4
36 Fetching: tunnel-vmc-plugin-0.2.2.gem (100%)
37 Successfully installed tunnel-vmc-plugin-0.2.2
38 Fetching: vmc-0.5.0.gem (100%)
39 Successfully installed vmc-0.5.0
40 Parsing documentation for json_pure-1.7.7
41 Installing ri documentation for json_pure-1.7.7
42 Parsing documentation for multi_json-1.7.2
43 Installing ri documentation for multi_json-1.7.2
44 Parsing documentation for interact-0.5.2
45 Installing ri documentation for interact-0.5.2
46 Parsing documentation for multipart-post-1.2.0
47 Installing ri documentation for multipart-post-1.2.0
48 Parsing documentation for rubyzip-0.9.9
49 Installing ri documentation for rubyzip-0.9.9
50 Parsing documentation for cf-uaa-lib-1.3.10
51 Installing ri documentation for cf-uaa-lib-1.3.10
52 Parsing documentation for cfoundry-0.5.2
53 Installing ri documentation for cfoundry-0.5.2
54 Parsing documentation for clouseau-0.0.2
55 Installing ri documentation for clouseau-0.0.2
56 Parsing documentation for mothership-0.5.1
57 Installing ri documentation for mothership-0.5.1
58 Parsing documentation for manifests-vmc-plugin-0.6.2

```

Gambar 1.4.: E-mail persetujuan dan pemberitahuan *credentials*

```

59 Installing ri documentation for manifests-vmc-plugin -0.6.2
60 Parsing documentation for addressable -2.3.4
61 Installing ri documentation for addressable -2.3.4
62 Parsing documentation for caldecott-client -0.0.2
63 Installing ri documentation for caldecott-client -0.0.2
64 Parsing documentation for mime-types -1.23
65 Installing ri documentation for mime-types -1.23
66 Parsing documentation for rest-client -1.6.7
67 Installing ri documentation for rest-client -1.6.7
68 Parsing documentation for uuidtools -2.1.4
69 Installing ri documentation for uuidtools -2.1.4
70 Parsing documentation for tunnel-vmc-plugin -0.2.2
71 Installing ri documentation for tunnel-vmc-plugin -0.2.2
72 Parsing documentation for vmc -0.5.0
73 Installing ri documentation for vmc -0.5.0
74 Done installing documentation for json_pure, multi_json,
75 interact, multipart-post, rubyzip, cf-uaa-lib, cfoundry,
76 clouseau, mothership, manifests-vmc-plugin, addressable,
77 caldecott-client, mime-types, rest-client, uuidtools,
78 tunnel-vmc-plugin, vmc (13 sec).
79 17 gems installed
80 $

```

Listing 1.1: Instalasi vmc

Pesan peringatan berikut ini:

```
1 WARNING: You don't have /home/bdp/.gem/ruby/2.0.0/bin in your PATH,
2   gem executables will not run.
```

Listing 1.2: Peringatan setting PATH untuk vmc

terjadi karena path untuk *executable script* Ruby yang diinstall oleh vmc belum ditambahkan ke variabel lingkungan PATH. Edit file \$HOME/.bashrc dan tambahkan baris berikut⁶:

```
1 export PATH=$PATH:/home/bdp/.gem/ruby/2.0.0/bin
```

Listing 1.3: Peringatan setting PATH untuk vmc

Setelah itu, ketikkan di shell prompt Bash: `source ~/.bashrc`⁷.

Hasil dari instalasi vmc adalah sebagai berikut:

```
1 $ gem list
2
3 *** LOCAL GEMS ***
4
5 addressable (2.3.4)
6 bigdecimal (1.2.0)
7 caldecott-client (0.0.2)
8 cf-uaa-lib (1.3.10)
9 cfoundry (0.5.2)
10 clouseau (0.0.2)
11 interact (0.5.2)
12 io-console (0.4.2)
13 json (1.7.7)
14 json_pure (1.7.7)
15 manifests-vmc-plugin (0.6.2)
16 mime-types (1.23)
17 minitest (4.3.2)
18 mothership (0.5.1)
19 multi_json (1.7.2)
20 multipart-post (1.2.0)
21 psych (2.0.0)
22 rake (0.9.6)
23 rdoc (4.0.0)
24 rest-client (1.6.7)
25 rubyzip (0.9.9)
26 test-unit (2.0.0.0)
27 tunnel-vmc-plugin (0.2.2)
28 uuidtools (2.1.4)
29 vmc (0.5.0)
30 $
```

Listing 1.4: Hasil gem yang terinstall

Periksa dengan menjalankan opsi help dari vmc:

⁶Catatan: "/home/bdp/" adalah direktori \$HOME saya, silahkan sesuaikan dengan tempat anda

⁷Ini hanya untuk keperluan saat ini saja, setelah ini tidak perlu lagi karena setiap login sudah akan dibaca oleh Bash

```

1 $ vmc help
2 Showing basic command set. Run with 'help --all' to list all commands.
3
4 Getting Started
5   target [URL]   Set or display the target cloud, organization, and space
6   logout        Log out from the target
7   login [EMAIL]  Authenticate with the target
8   info          Display information on the current target, user, etc.
9
10 Applications
11   app [APP]      Show app information
12   apps          List your applications
13
14   Management
15     start APPS...   Start an application
16     delete APPS...  Delete an application
17     push [NAME]     Push an application, syncing changes if it exists
18     restart APPS... Stop and start an application
19     stop APPS...    Stop an application
20
21 Services
22   service SERVICE Show service information
23   services        List your service
24
25   Management
26     delete-service [SERVICE] Delete a service
27     bind-service [SERVICE] [APP] Bind a service to an application
28     create-service [OFFERING] [NAME] Create a service
29     unbind-service [SERVICE] [APP] Unbind a service from an application
30     tunnel [INSTANCE] [CLIENT] Create a local tunnel to a service.
31
32 Organizations
33   org [ORGANIZATION] Show organization information
34   delete-org [ORGANIZATION] Delete an organization
35   orgs              List available organizations
36   create-org [NAME] Create an organization
37
38 Spaces
39   delete-space SPACES... Delete a space and its contents
40   spaces [ORGANIZATION] List spaces in an organization
41   create-space [NAME] [ORGANIZATION] Create a space in an organization
42   space [SPACE] Show space information
43
44 Routes
45   routes          List routes in a space
46
47 Domains
48   domains [SPACE] List domains in a space
49   map-domain NAME Map a domain to an organization or space
50   unmap-domain DOMAIN Unmap a domain from an organization or space
51
52 Options:
53   --[no-]color      Use colorful output
54   --[no-]script     Shortcut for --quiet and --force
55   --debug           Print full stack trace (instead of crash log)
56   -V, --verbose     Print extra information
57   -f, --[no-]force  Skip interaction when possible
58   -h, --help        Show command usage
59   -m, --manifest FILE Path to manifest file to use
60   -q, --[no-]quiet  Simplify output format
61   -t, --trace       Show API traffic

```

```

62  -u, --proxy EMAIL      Run this command as another user (admin)
63  -v, --version          Print version number
64  $

```

Listing 1.5: Hasil opsi help dari vmc

1.7.3. Konfigurasi di Server Cloud

Pada dasarnya, yang diperlukan hanyalah mengubah target ke server cloud dari CF dan kemudian mengubah password.

```

1  $ vmc target
2  NoMethodError: undefined method 'target' for nil:NilClass
3  For more information, see ~/.vmc/crash
4  $

```

Listing 1.6: Mengubah target server - belum ada konfigurasi

Error di atas terjadi karena file konfigurasi belum dibuat. File konfigurasi tersimpan di direktori \$HOME/.vmc. Mengubah target dilakukan dengan membuat file *target* di direktori tersebut. Isi dari file target tersebut adalah server CloudFoundry, yaitu <https://api.cloudfoundry.com>. Setelah itu, jika dieksekusi lagi, hasilnya adalah sebagai berikut:

```

1  $ vmc target
2  target: https://api.cloudfoundry.com
3  $

```

Listing 1.7: Mengubah target server - setelah konfigurasi

Setelah itu, setiap kali kita akan melakukan berbagai proses yang melibatkan server ini, kita harus melakukan proses login terlebih dahulu:

```

1  $ vmc login
2  target: https://api.cloudfoundry.com
3
4  Email> bambangpdp@yahoo.com
5
6  Password> *****
7
8  Authenticating... OK
9  $ vmc info --all
10 Getting runtimes... OK
11 Getting frameworks... OK
12 Getting services... OK
13
14 VMware's Cloud Application Platform
15
16 target: https://api.cloudfoundry.com
17   version: 0.999
18   support: http://support.cloudfoundry.com
19
20 user: bambangpdp@yahoo.com
21
22 runtime  description
23 java    1.6.0_24
24 java7   1.7.0_04

```

```

25 node          0.4.12
26 node06        0.6.8
27 node08        0.8.2
28 ruby18        1.8.7p357
29 ruby19        1.9.2p180
30
31 framework      description
32 grails
33 java_web
34 lift
35 node
36 play
37 rack
38 rails3
39 sinatra
40 spring
41 standalone
42
43 service        version   provider  description
44 mongodb        2.0      core      MongoDB NoSQL database
45 mysql          5.1      core      MySQL database
46 postgresql     9.0      core      PostgreSQL database (vFabric)
47 rabbitmq       2.4      core      RabbitMQ message queue
48 redis          2.4      core      Redis key-value store
49 redis          2.2      core      Redis key-value store
50 redis          2.6      core      Redis key-value store
51 $
52
53
54 $ vmc login
55 target: https://api.cloudfoundry.com
56
57 Email> bambangpdp@yahoo.com
58
59 Password> *****
60
61 Authenticating... OK
62
63 $ vmc info --all
64 Getting runtimes... OK
65 Getting frameworks... OK
66 Getting services... OK
67
68 VMware's Cloud Application Platform
69
70 target: https://api.cloudfoundry.com
71   version: 0.999
72   support: http://support.cloudfoundry.com
73
74 user: bambangpdp@yahoo.com
75
76 runtime        description
77 java           1.6.0_24
78 java7          1.7.0_04
79 node           0.4.12
80 node06         0.6.8
81 node08         0.8.2
82 ruby18         1.8.7p357
83 ruby19         1.9.2p180
84
85 framework      description
86 grails

```

```

87 java_web
88 lift
89 node
90 play
91 rack
92 rails3
93 sinatra
94 spring
95 standalone
96
97 service      version  provider  description
98 mongodb      2.0      core      MongoDB NoSQL database
99 mysql         5.1      core      MySQL database
100 postgresql   9.0      core      PostgreSQL database (vFabric)
101 rabbitmq      2.4      core      RabbitMQ message queue
102 redis         2.4      core      Redis key-value store
103 redis         2.6      core      Redis key-value store
104 redis         2.2      core      Redis key-value store
105 $

```

Listing 1.8: Login ke server

Untuk mengubah password:

```

1 $ vmc passwd
2 New Password> *****
3
4 Verify Password> *****
5
6 Your password strength is: good
7 Changing password... OK
8 $

```

Listing 1.9: Mengubah password server

1.7.4. Instalasi dan Konfigurasi Node.js di Komputer Lokal

Node.js tersedia untuk Linux, Windows, Mac OS X, serta SunOS. Untuk versi Linux, kebanyakan distro sudah menyertakan paket Node.js, hanya saja ada banyak versi dari Node.js dan jika kita menggunakan manajemen paket dari distro Linux, kita hanya bisa menginstall 1 versi saja. Sebagai contoh, di Arch Linux, paket Node.js bisa diinstall dengan perintah “pacman -S nodejs” tetapi hanya pada versi resmi di repo Arch Linux (versi 0.10.5 pada tanggal 6 Mei 2013).

Langkah instalasi berikut ini adalah langkah untuk instalasi tanpa manajemen paket dari distro Linux.

- ✓ Ambil paket *binary executable* dari <http://nodejs/download> atau langsung ke <http://nodejs.org/dist/>. Versi yang digunakan disini adalah 0.10.5. Download file tersebut, kemudian simpan di direktori tertentu (terserah anda, dibuku ini diletakkan di \$HOME/master/nodejs).

```

1 20:30:31 - bdp@bdp-arch: ~/master/nodejs$ ls
2 total 4672
3 drwxr-xr-x  2 bdp bdp      4096 Apr 26 20:37 .
4 drwxr-xr-x 46 bdp bdp      4096 May  4 23:33 ..
5 -rw-r--r--  1 bdp bdp 4770726 Apr 24 03:34 node-v0.10.5-linux-x86.tar.gz

```

```
6 20:30:31 - bdp@bdp-arch: ~ / master / nodejs$
```

Listing 1.10: Hasil dari download Node.js

- ✓ Ekstrak ke direktori yang diinginkan. Node.js akan diinstall di direktori `$HOME/software`:

```
1 $ cd
2 $ cd software
3 $ tar -xzf ~/master/nodejs/node-v0.10.5-linux-x86.tar.gz
4 $ ln -s node-v0.10.5-linux-x86 nodejs
5 $ ls -la
6 ....
7 ....
8 lrwxrwxrwx 1 bdp bdp 22 May 1 07:58 nodejs -> node-v0.10.5-linux-x86
9 drwxr-xr-x 5 bdp bdp 4096 Apr 24 03:31 node-v0.10.5-linux-x86
10 ....
11 ....
12 $
```

Listing 1.11: Ekstraksi Node.js

- ✓ Konfigurasi variabel lingkungan. Sebaiknya disimpan pada suatu file (pada buku ini, konfigurasi akan disimpan di `$HOME/environment/nodejs`):

```
1 NODEJS_HOME=/home/bdp/software/nodejs
2
3 PATH=$PATH:$NODEJS_HOME/bin
4 MANPATH=$MANPATH:$NODEJS_HOME/share/man
5 LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$NODEJS_HOME/lib
6 C_INCLUDE_PATH=$C_INCLUDE_PATH:$NODEJS_HOME/include
7 CPLUS_INCLUDE_PATH=$CPLUS_INCLUDE_PATH:$NODEJS_HOME/include
8
9 export PATH
10 export MANPATH
11 export LD_LIBRARY_PATH
12 export C_INCLUDE_PATH
13 export CPLUS_INCLUDE_PATH
```

Listing 1.12: Konfigurasi variabel lingkungan Node.js

- ✓ Setiap akan menggunakan Node.js, yang diperlukan adalah men-source file konfigurasi tersebut: `source ~/environment/nodejs`.

1.8. Pengelolaan Aplikasi di Cloud

Aplikasi yang dibuat nantinya akan di-deploy ke server CF. Pada umumnya, developer akan melakukan proses untuk upload (*push*), menghapus (*delete*), serta memperbaharui (*update*) aplikasi di server. Jika belum memahami sintaksis JavaScript serta penggunaan npm, jangan kuatir. Tujuan dari bab ini hanya mengenalkan pengelolaan aplikasi di Cloud. Aspek lainnya akan dibahas di bab-bab berikutnya.

1.8.1. Push, Delete, Update Aplikasi

Pada pembahasan ini, akan diberikan contoh menggunakan dua kategori, yaitu dengan menggunakan *framework* (ExpressJS - <http://expressjs.com>) serta tanpa menggunakan *framework*.

1.8.2. Menggunakan Framework ExpressJS

```
1 $ npm install -g express
```

Listing 1.13: Instalasi ExpressJS menggunakan npm

Jika berhasil, maka kita bisa menggunakan perintah *express* untuk membuat rerangka aplikasi. Sintaksis penggunaan ExpressJS adalah sebagai berikut:

```
1 $ express --help
2
3 Usage: express [options]
4
5 Options:
6
7   -h, --help            output usage information
8   -V, --version          output the version number
9   -s, --sessions         add session support
10  -e, --ejs               add ejs engine support (defaults to jade)
11  -J, --jshtml            add jshtml engine support (defaults to jade)
12  -H, --hogan             add hogan.js engine support
13  -c, --css <engine>     add stylesheet <engine> support (less|stylus) (defaults to plain css)
14  -f, --force            force on non-empty directory
15
16 $
```

Listing 1.14: Perintah express

Setelah itu, kita bisa membuat rerangka aplikasi ExpressJS dengan cara berikut:

```
1 $ mkdir hello
2 $ cd hello
3 $ express
4
5   create : .
6   create : ./package.json
7   create : ./app.js
8   create : ./public
9   create : ./public/javascripts
10  create : ./routes
11  create : ./routes/index.js
12  create : ./routes/user.js
13  create : ./public/stylesheet
14  create : ./public/stylesheet/style.css
15  create : ./views
16  create : ./views/layout.jade
17  create : ./views/index.jade
18  create : ./public/images
19
20  install dependencies:
21    $ cd . && npm install
22
23  run the app:
```

```
24 $ node app
25
26 $
```

Listing 1.15: Menggunakan express untuk membuat rerangka aplikasi

Pada rerangka aplikasi tersebut, terdapat file *package.json* untuk mendefinisikan aplikasi serta dependensi-nya dan *app.js* yang merupakan file utama untuk dijalankan pada server.

```
1 {
2   "name": "hello-node",
3   "version": "0.0.1",
4   "private": true,
5   "scripts": {
6     "start": "node app.js"
7   },
8   "dependencies": {
9     "express": "3.2.2",
10    "jade": "*"
11  }
12 }
```

Listing 1.16: package.json untuk ExpressJS

```
a
1
2 /**
3  * Module dependencies.
4  */
5
6 var express = require('express')
7   , routes = require('./routes')
8   , user = require('./routes/user')
9   , http = require('http')
10   , path = require('path');
11
12 var app = express();
13
14 // all environments
15 app.set('port', process.env.PORT || 3000);
16 app.set('views', __dirname + '/views');
17 app.set('view engine', 'jade');
18 app.use(express.favicon());
19 app.use(express.logger('dev'));
20 app.use(express.bodyParser());
21 app.use(express.methodOverride());
22 app.use(app.router);
23 app.use(express.static(path.join(__dirname, 'public')));
24
25 // development only
26 if ('development' == app.get('env')) {
27   app.use(express.errorHandler());
28 }
29
30 app.get('/', routes.index);
31 app.get('/users', user.list);
32
33 http.createServer(app).listen(app.get('port'), function(){
34   console.log('Express server listening on port ' + app.get('port'));
```

```
35 });
```

Listing 1.17: app.js untuk ExpressJS

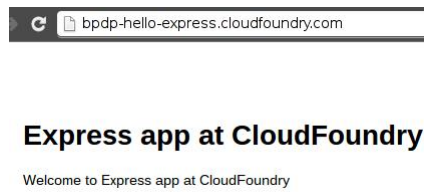
Edit file *routes/index.js* sebagai berikut:

```
1
2 /*
3  * GET home page.
4  */
5
6 exports.index = function(req, res){
7   res.render('index', { title: 'Express app at CloudFoundry' });
8 };
```

Listing 1.18: Hasil edit routes/index.js

Setelah itu, install modul-modul yang diperlukan dengan perintah *npm install* pada direktori tersebut. npm akan membaca file package.json kemudian menginstall modul-modul sesuai dengan deskripsi pada *dependencies*. Setelah diuji pada komputer lokal dengan perintah *node app.js*, dan sukses bisa diakses di browser dengan alamat *http://localhost:3000*, maka aplikasi tersebut bisa di-deploy di CloudFoundry. Proses deployment digambarkan sebagai berikut (anda sudah harus login menggunakan perintah *vmc login* sebelumnya) dan berada di direktori tempat aplikasi tersebut berada:

```
1 $ vmc push
2 Name> bdp-hello-express
3
4 Instances> 1
5
6 1: node
7 2: other
8 Framework> node
9
10 1: node
11 2: node06
12 3: node08
13 4: other
14 Runtime> 3
15
16 1: 64M
17 2: 128M
18 3: 256M
19 4: 512M
20 5: 1G
21 6: 2G
22 Memory Limit> 64M
23
24 Creating bdp-hello-express ... OK
25
26 1: bdp-hello-express.cloudfoundry.com
27 2: none
28 Domain> bdp-hello-express.cloudfoundry.com
29
30 Updating bdp-hello-express ... OK
31
32 Create services for application?> n
33
34 Save configuration?> n
35
```



Gambar 1.5.: Hasil push ke server

```

36 Uploading bdpd-hello-express... OK
37 Starting bdpd-hello-express... OK
38 Checking bdpd-hello-express...
39 1/1 instances: 1 running
40 OK
41 $

```

Listing 1.19: Deployment aplikasi ExpressJS ke CF

Hasilnya terlihat pada tampilan browser di Gambar 1.5

Aplikasi yang sudah dibuat seringkali diubah, oleh karena itu vmc juga menyediakan fasilitas untuk Mengupdate aplikasi.

```

1
2 /*
3  * GET home page.
4  */
5
6 exports.index = function(req, res){
7   var nv = process.version;
8   res.render('index', { title: 'Express app at CloudFoundry with Node.js ' + nv });
9 };

```

Listing 1.20: Update: menambahkan versi Node.js ke routes/index.js

```

1 $ vmc apps
2 Getting applications... OK
3
4 name          status    usage    runtime    url
5 hello-express running    1 x 64M   node08     bdpd-hello-express.cloudfoundry.com
6 $ vmc push hello-express
7 Uploading hello-express... OK
8 Stopping hello-express... OK
9
10 Starting hello-express... OK
11 Checking hello-express... OK
12 $

```

Listing 1.21: Mengupdate aplikasi di server

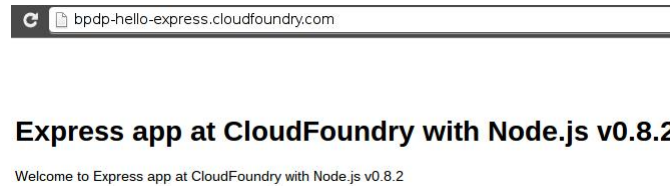
Hasilnya bisa dilihat di Gambar 1.6

Untuk menghapus aplikasi:

```

1 $ vmc delete bdpd-m1-hellonoframework
2 Really delete bdpd-m1-hellonoframework?> y
3

```



Gambar 1.6.: Hasil update dengan menyertakan versi Node.js

```
4 Deleting bmdp-m1-hellonoframework ... OK
5
6 $
```

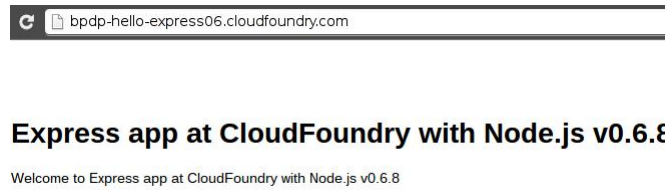
Listing 1.22: Menghapus aplikasi yang di-deploy di CF

Pada saat deployment, kita juga bisa memilih versi Node.js (runtime) sebagai berikut:

```
1 $ vmc push --runtime=node08
2 Name> bmdp-hello-express06
3
4 Instances> 1
5
6 1: node
7 2: other
8 Framework> node
9
10 1: 64M
11 2: 128M
12 3: 256M
13 4: 512M
14 5: 1G
15 Memory Limit> 64M
16
17 Creating bmdp-hello-express06 ... OK
18
19 1: bmdp-hello-express06.cloudfoundry.com
20 2: none
21 URL> bmdp-hello-express06.cloudfoundry.com
22
23 Updating bmdp-hello-express06 ... OK
24
25 Create services for application?> n
26
27 Save configuration?> n
28
29 Uploading bmdp-hello-express06 ... OK
30 Starting bmdp-hello-express06 ... OK
31 Checking bmdp-hello-express06 ... OK
32 $
```

Listing 1.23: Deployment ke CF dengan memilih runtime Node.js

Hasilnya bisa dilihat di Gambar 1.7



Gambar 1.7.: Deployment menggunakan versi runtime tertentu

1.8.3. Tanpa Framework

Tanpa *framework*, yang kita perlukan hanyalah langsung mem-*push* file yang kita buat (dalam contoh ini adalah app.js):

```

1 var http = require('http');
2 var url = require("url");
3
4 http.createServer(function (req, res) {
5
6     var pathname = url.parse(req.url).pathname;
7
8     res.writeHead(200, {'Content-Type': 'text/html'});
9     res.write("Hello NodeJS <u>" + process.version + "</u>");
10    res.write("<br />Request for <b>" + pathname + "</b> received.");
11    res.end();
12
13 }).listen(1337);

```

Listing 1.24: app.js tanpa framework

Proses deployment adalah sebagai berikut:

```

1 $ vmc push
2 Name> hello-noframework
3
4 Instances> 1
5
6 1: node
7 2: other
8 Framework> node
9
10 1: node
11 2: node06
12 3: node08
13 4: other
14 Runtime> 3
15
16 1: 64M
17 2: 128M
18 3: 256M
19 4: 512M
20 5: 1G
21 Memory Limit> 64M
22
23 Creating hello-noframework... OK
24
25 1: hello-noframework.cloudfoundry.com

```



Gambar 1.8.: Hasil deployment app.js tanpa framework

```
26 2: none
27 URL> hello-noframework.cloudfoundry.com
28
29 Updating hello-noframework... OK
30
31 Create services for application?> n
32
33 Save configuration?> n
34
35 Uploading hello-noframework... OK
36 Starting hello-noframework... OK
37 Checking hello-noframework... OK
38 $
```

Listing 1.25: Deployment app.js tanpa framework

Hasilnya bisa dilihat pada Gambar 1.8

REPL dan Dasar-dasar JavaScript di Node.js

2.1. REPL

REPL adalah lingkungan pemrograman interaktif, tempat developer bisa mengetikkan program per baris dan langsung mengeksekusi hasilnya. Biasanya ini digunakan untuk menguji perintah-perintah yang cukup dijalankan pada satu baris atau satu blok segmen kode sumber saja. Karena fungsinya itu, maka istilah yang digunakan adalah REPL (read-eval-print-loop), yaitu loop atau perulangan baca perintah - evaluasi perintah - tampilkan hasil. REPL sering juga disebut sebagai *interactive top level* atau *language shell*. “Tradisi” ini sudah dimulai sejak jaman LISP di mesin UNIX di era awal pengembangan *development tools*. Saat ini hampir semua *interpreter/compiler* mempunyai REPL, misalnya Python, Ruby, Scala, PHP, berbagai interpreter/compiler LISP, dan tidak ketinggalan Node.js.

2.1.1. Mengaktifkan REPL

Untuk mengaktifkan REPL dari Node.js, *executable command line program*-nya adalah **node**. Jika **node** dipanggil dengan argumen nama file JavaScript, maka file JavaScript tersebut akan dieksekusi, sementara jika tanpa argumen, akan masuk ke REPL:

```
1 $ node
2 > .help
3 .break Sometimes you get stuck, this gets you out
4 .clear Alias for .break
5 .exit Exit the repl
6 .help Show repl options
7 .load Load JS from a file into the REPL session
8 .save Save all evaluated commands in this REPL session to a file
9 >
```

Listing 2.1: Node.js REPL

Tanda “>” adalah tanda bahwa REPL Node.js siap untuk menerima perintah. Untuk melihat perintah-perintah REPL, bisa digunakan **.help**.

2.1.2. Perintah-perintah REPL

Pada sesi REPL, kita bisa memberikan perintah internal REPL maupun perintah-perintah lain yang sesuai dan dikenali sebagai perintah JavaScript. Perintah internal REPL Node.js terdiri atas:

- ✓ **.break**: keluar dan melepaskan diri dari "keruwetan" baris perintah di REPL.
- ✓ **.clear**: alias untuk **.break**
- ✓ **.exit**: keluar dari sesi REPL (bisa juga dengan menggunakan Ctrl-D)
- ✓ **.help**: menampilkan pertolong perintah internal REPL
- ✓ **.load**: membaca dan mengeksekusi perintah-perintah JavaScript yang terdapat pada suatu file.
- ✓ **.save**: menyimpan sesi REPL ke dalam suatu file.

Contoh untuk **.load**:

```
1 $ node
2 > .load /home/bpdp/kerjaan/src/javascript/nodejs/hello.js
3 > var http = require('http');
4 undefined
5 > http.createServer(function (req, res) {
6 ...   res.writeHead(200, {'Content-Type': 'text/plain'});
7 ...   res.end('Hello World\n');
8 ... }).listen(1337, '127.0.0.1');
9 { domain: null,
10   _events:
11     { request: [Function],
12       connection: [Function: connectionListener],
13       clientError: [Function] },
14   _maxListeners: 10,
15   _connections: 0,
16   connections: [Getter/Setter],
17   _handle: null,
18   _usingSlaves: false,
19   _slaves: [],
20   allowHalfOpen: true,
21   httpAllowHalfOpen: false,
22   timeout: 120000 }
23 > console.log('Server running at http://127.0.0.1:1337/');
24 Server running at http://127.0.0.1:1337/
25 undefined
26 >
```

Listing 2.2: Contoh penggunaan **.load** dalam REPL

Setelah keluar dari sesi REPL, maka port akan ditutup dan hasil eksekusi di atas akan dibatalkan.

Untuk menyimpan hasil sesi REPL menggunakan **.save**, jika tanpa menyebutkan direktori, maka akan disimpan di direktori aktif saat itu. Contoh:

```
1 $ node
2 > console.log("Selamat datang di Node.js")
3 Selamat datang di Node.js
4 undefined
5 > .save /home/bdp/kerjaan/src/javascript/nodejs/welcome.js
6 Session saved to:/home/bdp/kerjaan/src/javascript/nodejs/welcome.js
7 > $ cat /home/bdp/kerjaan/src/javascript/nodejs/welcome.js
8 console.log("Selamat datang di Node.js")
9 $
```

Listing 2.3: Contoh penggunaan perintah .save di sesi REPL

2.2. Dasar-dasar JavaScript di Node.js

Node.js merupakan sistem peranti lunak yang merupakan implementasi dari bahasa pemrograman JavaScript. Spesifikasi JavaScript yang diimplementasikan merupakan spesifikasi resmi dari ECMAScript serta CommonJS (<http://commonjs.org>). Dengan demikian, jika anda sudah pernah mempelajari JavaScript sebelumnya, tata bahasa dari perintah yang dipahami oleh Node.js masih tetap sama dengan JavaScript.

2.2.1. Membaca *Masukan* dari Stream / Masukan Standar (stdin)

Untuk lebih memahami dasar-dasar JavaScript serta penerapannya di Node.js, seringkali kita perlu melakukan simulasi pertanyaan - proses - keluaran jawaban. Proses akan kita pelajari seiring dengan materi-materi berikutnya, sementara untuk keluaran, kita bisa menggunakan **console.log**. Bagian ini akan menjelaskan sedikit tentang masukan.

Perintah untuk memberi masukan di Node.js sudah tersedia pada pustaka API *Readline*¹. Pola dari masukan ini adalah sebagai berikut:

- ✓ me-require pustaka Readline
- ✓ membuat *interface* untuk masukan dan keluaran
- ✓ .. gunakan interface ..
- ✓ .. gunakan interface ..
- ✓ .. gunakan interface ..
- ✓ .. gunakan interface ..
- ✓ ..
- ✓ ..
- ✓ tutup *interface*

¹Lengkapannya bisa diakses di <http://nodejs.org/api/readline.html>

Implementasi dari pola diatas bisa dilihat pada kode sumber berikut ini (diambil dari manual Node.js):

```
1 var readline = require('readline');
2
3 var rl = readline.createInterface({
4   input: process.stdin,
5   output: process.stdout
6 });
7
8 rl.question("What do you think of node.js? ", function(answer) {
9   console.log("Thank you for your valuable feedback:", answer);
10  rl.close();
11 });
12
13 // hasil:
14 // $ node readline.js
15 // What do you think of node.js? awesome!
16 // Thank you for your valuable feedback: awesome!
17 // $
```

Listing 2.4: readline.js: penggunaan pustaka Readline untuk masukan

Catatan: *function(answer)* pada listing di atas merupakan *anonymous function* atau fungsi anonim (sering juga disebut *lambda function* / fungsi lambda. Posisi fungsi pada listing tersebut disebut dengan fungsi *callback*. Untuk keperluan pembahasan saat ini, untuk sementara yang perlu dipahami adalah hasil input akan dimasukkan ke *answer* untuk diproses lebih lanjut. Fungsi dan *callback* akan dibahas lebih lanjut pada pembahasan berikutnya.

2.2.2. Nilai/Value dan Tipe Data

Program dalam JavaScript akan berhubungan dengan data atau nilai. Setiap nilai mempunyai tipe tertentu. JavaScript mengenali berbagai tipe berikut ini:

- ✓ Angka: bulat (misalnya 4) atau pecahan (misalnya 3.75)
- ✓ *Boolean*: nilai benar (true) dan salah (false)
- ✓ String: diapit oleh tanda petik ganda ("contoh string") atau tunggal ('contoh string')
- ✓ *null*
- ✓ *undefined*

JavaScript adalah bahasa pemrograman yang memungkinkan pemrogram untuk tidak mendefinisikan tipe data pada saat deklarasi, atau sering juga disebut sebagai *dynamically typed language*:

```
1 var jumlahMahasiswa = 30
2 console.log('Jumlah mahasiswa dalam satu kelas = ' + jumlahMahasiswa);
3 // Jumlah mahasiswa dalam satu kelas = 30
```

Listing 2.5: Fitur *dynamically typed language*

Pada contoh di atas, kita bisa melihat bahwa data akan dikonversi secara otomatis pada saat program dieksekusi.

Catatan:

- ✓ Khusus untuk operator "+", JavaScript akan melakukan penggabungan string (*string concatenation*), tetapi untuk operator lain, akan dilakukan operasi matematis sesuai operator tersebut (-,/,*).
- ✓ Konversi string ke tipe numerik bisa dilakukan dengan *parseInt(string)* (jika bilangan bulat) dan *parseFloat(string)* (jika bilangan pecahan).

2.2.3. Variabel

Variabel adalah suatu nama yang didefinisikan untuk menampung suatu nilai. Nama ini akan digunakan sebagai referensi yang akan menunjukkan ke nilai yang ditampungnya. Nama variabel disebut dengan *identifier* / pengenal. Ada beberapa syarat pemberian nama *identifier* di JavaScript:

- ✓ Dimulai dengan huruf, *underscore* (`_`), atau tanda dollar (`$`).
- ✓ Karakter berikutnya bisa berupa angka, selain ketentuan pertama di atas.
- ✓ Membedakan huruf besar - kecil.

Konvensi yang digunakan oleh pemrogram JavaScript terkait dengan penamaan ini adalah variasi dari metode *camel case*, yaitu *camelBack*. Contoh: `jumlahMahasiswa`, `linkMenu`, `status`.

2.2.4. Konstanta

Konstanta mirip dengan variabel, hanya saja sifatnya *read-only*, tidak bisa diubah-ubah setelah ditetapkan. Untuk menetapkan konstanta di JavaScript, digunakan kata kunci *const*. Contoh:

```
1 const MENU = "Home";
2
3 console.log("Posisi menu = " + MENU);
4
5 // mencoba mengisi MENU. berhasil?
6
7 MENU = "About";
8
```

```

9 console.log("Posisi menu = " + MENU);
10
11 // Posisi menu = Home
12 // Posisi menu = Home

```

Listing 2.6: Contoh konstanta dalam JavaScript

Konvensi penamaan konstanta adalah menggunakan huruf besar semua. Bagian ini (sampai saat buku ini ditulis) hanya berlaku di Firefox dan Google Chrome - V8 (artinya berlaku juga untuk Node.js).

2.2.5. Fungsi

Pengertian Fungsi

Fungsi merupakan subprogram atau suatu bagian dari keseluruhan program yang ditujukan untuk mengerjakan suatu pekerjaan tertentu dan (biasanya) menghasilkan suatu nilai kembalian. Subprogram ini relatif independen terhadap bagian-bagian lain sehingga memenuhi kaidah "bisa-digunakan-kembali" atau *reusable* pada beberapa program yang memerlukan fungsionalitasnya. Fungsi dalam ilmu komputer sering kali juga disebut dengan *prcedure*, *routine*, atau *method*.

Definisi Fungsi

Definisi fungsi dari JavaScript di Node.js bisa dilakukan dengan sintaksis berikut ini:

```

1 function namaFungsi(argumen1, argumen2, ... , argumenN) {
2   ..
3   JavaScript code ..
4   JavaScript code ..
5   JavaScript code ..
6   JavaScript code ..
7   ..
8 }

```

Listing 2.7: Sintaksis Fungsi dalam JavaScript

Setelah dideklarasikan, fungsi tersebut bisa dipanggil dengan cara sebagai berikut:

```

1 ..
2 ..
3 namaFungsi(argumen1, argumen2, ... , argumenN);
4 ..
5 ..

```

Listing 2.8: Pemanggilan Fungsi dalam JavaScript

Contoh dalam program serta pemanggilannya adalah sebagai berikut:

```

1 $ node
2 > function addX(angka) {
3   ... console.log(angka + 10);
4   ... }
5 undefined
6 > addX(20);
7 30

```

```
8 undefined
9 >
10 > function add2Numbers(angka1, angka2) {
11 ... return angka1 + angka2;
12 ... }
13 undefined
14 > console.log("232 + 432 = " + add2Numbers(232, 432));
15 232 + 432 = 664
16 undefined
17 >
```

Listing 2.9: Contoh deklarasi fungsi dan pemanggilannya

Fungsi Anonim

Fungsi anonim adalah fungsi tanpa nama, pemrogram tidak perlu memberikan nama ke fungsi. Biasanya fungsi anonim ini hanya digunakan untuk fungsi yang dikerjakan pada suatu bagian program saja dan tidak dengan maksud untuk dijadikan komponen yang bisa dipakai di bagian lain dari program (biasanya untuk menangani *event* atau *callback*). Untuk mendeklarasikan fungsi ini, digunakan literal *function*.

```
1 var pangkat = function(angka) {return angka * angka};
2 console.log(pangkat(10));
3 // output: 100
```

Listing 2.10: Fungsi anonim

Fungsi Rekursif

Fungsi rekursif adalah fungsi yang memanggil dirinya sendiri. Contoh dari aplikasi fungsi rekursif adalah pada penghitungan faktorial berikut:

```
1 function factorial(n) {
2
3     if ((n == 0) || (n == 1))
4         return 1;
5     else
6         return (n * factorial(n - 1));
7
8 }
9
10 console.log("factorial(6) = " + factorial(6));
11
12 // hasil:
13 // factorial(6) = 720
```

Listing 2.11: Fungsi rekursif untuk menghitung faktorial

Fungsi di dalam Fungsi / *Nested Functions*

Saat mendefinisikan fungsi, di dalam fungsi tersebut pemrogram bisa mendefinisikan fungsi lainnya. Meskipun demikian, fungsi yang terletak dalam suatu definisi fungsi tidak bisa diakses dari luar fungsi tersebut dan hanya tersedia untuk fungsi yang didefinisikan.

```
1 function induk() {
2
3   var awal = 0;
4   function tambahkan() {
5     awal++;
6   }
7
8   tambahkan();
9   tambahkan();
10
11   console.log('Nilai = ' + awal);
12
13 }
14
15 induk();
16 tambahkan();
17
18 // hasil:
19 // Nilai = 2
20 //
21 // /home/bpdp/kerjaan/git-repos/buku-cloud-nodejs/
22 // src/bab-02/nested.js:12
23 // tambahkan();
24 // ~
25 // ReferenceError: tambahkan is not defined
26 //   at Object.<anonymous> (/home/bpdp/kerjaan/git-repos/
27 //     buku-cloud-nodejs/src/bab-02/nested.js:12:1)
28 //   at Module._compile (module.js:456:26)
29 //   at Object.Module._extensions..js (module.js:474:10)
30 //   at Module.load (module.js:356:32)
31 //   at Function.Module._load (module.js:312:12)
32 //   at Function.Module.runMain (module.js:497:10)
33 //   at startup (node.js:119:16)
34 //   at node.js:901:3
```

Listing 2.12: Fungsi di dalam Fungsi

2.2.6. Literal

Literal digunakan untuk merepresentasikan nilai dalam JavaScript. Ada beberapa tipe literal.

Literal Array

Array atau variabel berindeks adalah penampung untuk obyek yang menyerupai *list* atau daftar. Obyek array juga menyediakan berbagai fungsi dan metode untuk mengolah anggota yang terdapat dalam daftar tersebut (terutama untuk operasi *traversal* dan permutasi. Listing berikut menunjukkan beberapa operasi untuk literal array.

```
1 var arrMembers = ['one', 'two', , 'three'];
2 // sengaja ada koma di bagian akhir
3 console.log(arrMembers[0]);
4 // hasil: one
5 console.log(arrMembers[2]);
6 // hasil: undefined
7 console.log(arrMembers[3]);
8 // hasil: three
```

```
9 console.log(arrMembers[4]);
10 // hasil: undefined - karena tidak ada
11 console.log(arrMembers.length);
12 // hasil: 4
13 var multiArray = [
14     ['0-0', '0-1', '0-2'],
15     ['1-0', '1-1', '1-2'],
16     ['2-0', '2-1', '2-2']];
17 console.log(multiArray[0][2]);
18 // hasil: 0-2
19 console.log(multiArray[1][2]);
20 // hasil: 1-2
```

Listing 2.13: Array di JavaScript

Literal Boolean

Literal boolean menunjukkan nilai benar (true) atau salah (false).

Literal Integer

Literal integer digunakan untuk mengekspresikan nilai bilangan bulat. Nilai bulangan bulat dalam JavaScript bisa dalam bentuk:

- ✓ decimal (basis 10): digit tanpa awalan nol.
- ✓ octal (basis 8): digit diawali dengan 1 angka nol.²
- ✓ hexadecimal (basis 16): digit diawali dengan 0x.

Literal Floating-point

Literal ini digunakan untuk mengekspresikan nilai bilangan pecahan, misalnya 0.4343 atau bisa juga menggunakan E/e (nilai eksponensial), misalnya -3.1E12.

Literal Obyek

Literal ini akan dibahas di bab yang menjelaskan tentang paradigma pemrograman berorientasi obyek di JavaScript.

Literal String

Literal string mengekspresikan suatu nilai dalam bentuk sederetan karakter dan berada dalam tanda petik (ganda/“” maupun tunggal/”). Contoh:

- ✓ “Kembali ke halaman utama”
- ✓ 'Lisensi'
- ✓ “Hari ini, Jum’at, tanggal 21 November”

²pada ECMA-262, bilangan octal ini sudah tidak digunakan lagi.

- ✓ "1234.543"
- ✓ "baris pertama \n baris kedua"

Contoh terakhir di atas menggunakan karakter khusus (`\n`). Beberapa karakter khusus lainnya adalah:

- ✓ `\b`: Backspace
- ✓ `\f`: Form feed
- ✓ `\n`: New line
- ✓ `\r`: Carriage return
- ✓ `\t`: Tab
- ✓ `\v`: Vertical tab
- ✓ `\'`: Apostrophe atau single quote
- ✓ `\"`: Double quote
- ✓ `\\`: Backslash (`\`).
- ✓ `\XXX`: Karakter dengan pengkodean Latin-1 dengan tiga digit octal antara 0 and 377. (misal, `\251` adalah simbol hak cipta).
- ✓ `\xXX`: seperti di atas, tetapi hexadecimal (2 digit).
- ✓ `\uXXXX`: Karakter *Unicode* dengan 3 digit karakter hexadecimal.

Backslash sendiri sering digunakan sebagai *escape character*, misalnya "NaN sering disebut juga sebagai `\`'Not a Number`\`'".

2.2.7. Struktur Data dan Representasi JSON

JSON (*JavaScript Object Notation*) adalah subset dari JavaScript dan merupakan struktur data native di JavaScript. Bentuk dari representasi struktur data JSON adalah sebagai berikut³:

```
1 var data = {  
2     "firstName": "John",  
3     "lastName": "Smith",  
4     "age": 25,  
5     "address": {  
6         "streetAddress": "21 2nd Street",  
7         "city": "New York",  
8         "state": "NY",  
9         "postalCode": "10021"  
10    },  
}
```

³<http://en.wikipedia.org/wiki/JSON> dengan sedikit perubahan

```

11     "phoneNumber":
12     {
13         "home": "212 555-1234",
14         "fax": "646 555-4567"
15     }
16 }
17
18 console.log(data.firstName + " " + data.lastName +
19     " has this phone number = "
20     + data.phoneNumber.home );
21
22 // hasil:
23 // John Smith has this phone number = 212 555-1234

```

Listing 2.14: Representasi *JSON*

Dari representasi di atas, kita bisa membaca:

- ✓ Nilai data “firstname” adalah “John”
- ✓ Data “address” terdiri atas sub data “streetAddress”, “city”, “state”, dan “postal-Code” yang masing-masing mempunyai nilai data sendiri-sendiri.
- ✓ dan seterusnya

2.2.8. Aliran Kendali

Alur program dikendalikan melalui pernyataan-pernyataan untuk aliran kendali. Ada beberapa pernyataan aliran kendali yang akan dibahas.

Pernyataan Kondisi *if .. else if .. else*

Pernyataan ini digunakan untuk mengerjakan atau tidak mengerjakan suatu bagian atau blok program berdasarkan hasil evaluasi kondisi tertentu.

```

1  var kondisi = false;
2  if (kondisi) {
3      console.log('hanya dikerjakan jika kondisi bernilai benar/true');
4  };
5  // hasil: n/a, tidak ada hasilnya
6  var kondisi = true;
7  if (kondisi) {
8      console.log('hanya dikerjakan jika kondisi bernilai benar/true');
9  };
10 // hasil: hanya dikerjakan jika kondisi bernilai benar/true
11 // Contoh berikut lebih kompleks, melibatkan input
12
13 var readline = require('readline');
14
15 var rl = readline.createInterface({
16     input: process.stdin,
17     output: process.stdout
18 });
19
20 rl.question("Masukkan angka nilai: ", function(answer) {
21     if (answer > 80) {
22         console.log("Nilai: A");

```

```

23 } else if (answer > 70) {
24   console.log("Nilai: B");
25 } else if (answer > 40) {
26   console.log("Nilai: C");
27 } else if (answer > 30) {
28   console.log("Nilai: D");
29 } else {
30   console.log("Tidak lulus");
31 }
32 rl.close();
33 });
34
35 // hasil:
36 // hanya dikerjakan jika kondisi bernilai benar/true
37 // Masukkan angka nilai: 50
38 // Nilai: C

```

Listing 2.15: Pernyataan *if .. else if .. else*

Pernyataan *switch*

Pernyataan ini digunakan untuk mengevaluasi suatu ekspresi dan membandingkan sama atau tidaknya dengan suatu label tertentu di dalam struktur pernyataan switch, serta mengeksekusi perintah-perintah sesuai dengan label yang cocok.

```

1  var readline = require('readline');
2
3  var rl = readline.createInterface({
4    input: process.stdin,
5    output: process.stdout
6  });
7
8  console.log("Menu");
9  console.log("====");
10 console.log("1. Mengisi data");
11 console.log("2. Mengedit data");
12 console.log("3. Menghapus data");
13 console.log("4. Mencari data");
14 rl.question("Masukkan angka pilihan anda: ", function(answer) {
15   console.log("Pilihan anda: " + answer);
16   switch (answer) {
17     case "1":
18       console.log("Anda memilih menu pengisian data");
19       break;
20     case "2":
21       console.log("Anda memilih menu pengeditan data");
22       break;
23     case "3":
24       console.log("Anda memilih menu penghapusan data");
25       break;
26     case "4":
27       console.log("Anda memilih menu pencarian data");
28       break;
29     default:
30       console.log("Anda tidak memilih salah satu dari menu di atas");
31       break;
32   }
33   rl.close();
34 });
35

```

```
36 // hasil:
37 // $ node switch.js
38 // Menu
39 // ====
40 // 1. Mengisi data
41 // 2. Mengedit data
42 // 3. Menghapus data
43 // 4. Mencari data
44 // Masukkan angka pilihan anda: 10
45 // Pilihan anda: 10
46 // Anda tidak memilih salah satu dari menu di atas
47 // $ node switch.js
48 // Menu
49 // ====
50 // 1. Mengisi data
51 // 2. Mengedit data
52 // 3. Menghapus data
53 // 4. Mencari data
54 // Masukkan angka pilihan anda: 2
55 // Pilihan anda: 2
56 // Anda memilih menu pengeditan data
```

Listing 2.16: Pernyataan *Contoh penggunaan “switch”*

Looping

Looping atau sering juga disebut “kalang” adalah konstruksi program yang digunakan untuk melakukan suatu blok perintah secara berulang-ulang.

for

```
1 for (var i = 0; i < 9; i++) {
2   console.log(i);
3 }
4
5 // hasil:
6 // 0
7 // 1
8 // 2
9 // 3
10 // 4
11 // 5
12 // 6
13 // 7
14 // 8
```

Listing 2.17: Pernyataan *for*

Pernyataan “for” juga bisa digunakan untuk mengakses data yang tersimpan dalam struktur data JavaScript (JSON).

```
1 var data = {a:1, b:2, c:3};
2
3 for (var iterasi in data) {
4   console.log("Nilai dari iterasi " + iterasi + " adalah: " + data[iterasi]);
5 }
6
7 // hasil:
8 // Nilai dari iterasi a adalah: 1
9 // Nilai dari iterasi b adalah: 2
```

```
10 // Nilai dari iterasi c adalah: 3
```

Listing 2.18: Pernyataan *for .. in*

do .. while

Pernyataan ini digunakan untuk mengerjakan suatu blok program selama suatu kondisi bernilai benar dengan jumlah minimal pengerjaan sebanyak 1 kali.

```
1 var i = 0;
2 do {
3   i += 2;
4   console.log(i);
5 } while (i < 20);
6
7 // hasil:
8 // 2
9 // 4
10 // 6
11 // 8
12 // 10
13 // 12
14 // 14
15 // 16
16 // 18
17 // 20
```

Listing 2.19: Pernyataan *do .. while*

while

Seperti *do .. while*, pernyataan ini digunakan untuk mengerjakan suatu blok program secara berulang-ulang selama kondisi bernilai benar. Meskipun demikian, bisa saja blok program tersebut tidak pernah dikerjakan jika pada saat awal ekspresi dievaluasi sudah bernilai *false*.

```
1 var n = 0;
2 var x = 0;
3
4 while (n < 5) {
5   n++;
6   x += n;
7   console.log("Nilai n = " + n);
8   console.log("Nilai x = " + x);
9 }
10
11 // hasil:
12 // Nilai n = 1
13 // Nilai x = 1
14 // Nilai n = 2
15 // Nilai x = 3
16 // Nilai n = 3
17 // Nilai x = 6
18 // Nilai n = 4
19 // Nilai x = 10
20 // Nilai n = 5
21 // Nilai x = 15
```

Listing 2.20: Pernyataan *while*

label, break, dan continue

Bagian ini digunakan dalam *looping* dan *switch*.

- ✓ *label* digunakan untuk memberi pengenalan pada suatu lokasi program sehingga bisa direferensi oleh *break* maupun *continue* (jika dikehendaki).
- ✓ *break* digunakan untuk menghentikan eksekusi dan meneruskan alur program ke pernyataan setelah *looping* atau *switch*
- ✓ *continue* digunakan untuk meneruskan eksekusi ke iterasi atau ke kondisi *switch* berikutnya.

```
1 var n = 0;
2 var x = 0;
3
4 while (n < 5) {
5     n++;
6     x += n;
7
8     if (x%2 == 0) {
9         continue;
10    };
11
12    if (x>10) {
13        break;
14    };
15
16    console.log("Nilai n = " + n);
17    console.log("Nilai x = " + x);
18
19 };
20 // hasil:
21 //Nilai n = 1
22 //Nilai x = 1
23 //Nilai n = 2
24 //Nilai x = 3
```

Listing 2.21: Pernyataan *break* dan *continue*

```
1 topLabel:
2   for(var k = 0; k < 10; k++){
3     for(var m = 0; m < 20; m++){
4       if(m == 5){
5         console.log("Nilai k = " + k);
6         console.log("Nilai m = " + m);
7         break topLabel;
8       }
9     }
10  }
11 // hasil:
12 //Nilai k = 0
13 //Nilai m = 5
```

Listing 2.22: Pernyataan *break* dengan label

2.2.9. Penanganan Error

JavaScript mendukung pernyataan *try .. catch .. finally* serta *throw* untuk menangani error. Meskipun demikian, banyak hal yang tidak sesuai dengan konstruksi ini karena sifat

JavaScript yang *asynchronous*. Untuk kasus asynchronous, pemrogram lebih disarankan menggunakan *function callback*.

```
1 try {
2   gakAdaFungsiIni();
3 } catch (e) {
4   console.log ("Error: " + e.message);
5 } finally {
6   console.log ("Bagian 'pembersihan', akan dikerjakan, apapun yang terjadi");
7 };
8
9 // hasil:
10 // Error: gakAdaFungsiIni is not defined
11 // Bagian 'pembersihan', akan dikerjakan, apapun yang terjadi
```

Listing 2.23: Pernyataan *try catch finally*

Jika diperlukan, kita bisa mendefinisikan sendiri error dengan menggunakan pernyataan *throw*.

```
1 try {
2   var a = 1/0;
3   throw "Pembagian oleh angka 0";
4 } catch (e) {
5   console.log ("Error: " + e);
6 };
7
8 // hasil:
9 // Error: Pembagian oleh angka 0
```

Listing 2.24: Pernyataan *try catch throw*

3.1. Pemrograman Fungsional

Pemrograman fungsional, atau sering disebut *functional programming*, selama ini lebih sering dibicarakan di level para akademisi. Meskipun demikian, saat ini terdapat kecenderungan paradigma ini semakin banyak digunakan di industri. Contoh nyata dari implementasi paradigma ini di industri antara lain adalah Scala (<http://www.scala-lang.org>), OCaml (<http://www.ocaml.org>), Haskell (<http://www.haskell.org>), Microsoft F# (<http://fsharp.org>), dan lain-lain. Dalam konteks paradigma pemrograman, peranti lunak yang dibangun menggunakan pendekatan paradigma ini akan terdiri atas berbagai fungsi yang mirip dengan fungsi matematis. Fungsi matematis tersebut dievaluasi dengan penekanan pada penghindaran *state* serta *mutable data*. Bandingkan dengan paradigma pemrograman prosedural yang menekankan pada *immutable data* dan definisi berbagai prosedur dan fungsi untuk mengubah *state* serta data.

JavaScript bukan merupakan bahasa pemrograman fungsional yang murni, tetapi ada banyak fitur dari pemrograman fungsional yang terdapat dalam JavaScript. Dalam hal ini, JavaScript banyak dipengaruhi oleh bahasa pemrograman Scheme (<http://www.schemers.org/>). Bab ini akan membahas beberapa fitur pemrograman fungsional di JavaScript. Pembahasan ini didasari pembahasan di bab sebelumnya tentang Fungsi di JavaScript.

3.1.1. Ekspresi Lambda

Ekspresi lambda / *lambda expression* merupakan hasil karya dari ALonzo Church sekitar tahun 1930-an. Aplikasi dari konsep ini di dalam pemrograman adalah penggunaan fungsi sebagai parameter untuk suatu fungsi. Dalam pemrograman, *lambda function* sering juga disebut sebagai fungsi anonim (fungsi yang dipanggil/dieksekusi tanpa

ditautkan (*bound*) ke suatu *identifier*). Berikut adalah implementasi dari konsep ini di JavaScript:

```
1 // Diambil dari
2 // http://stackoverflow.com/questions/3865335/what-is-a-lambda-language
3 // dengan beberapa perubahan
4
5 function applyOperation(a, b, operation) {
6     return operation(a, b);
7 }
8
9 function add(a, b) {
10     return a+b;
11 }
12
13 function subtract(a, b) {
14     return a-b;
15 }
16
17 console.log('1,2, add: ' + applyOperation(1,2, add));
18 console.log('43,21, subtract: ' + applyOperation(43,21, subtract));
19
20 console.log('4^3: ' + applyOperation(4, 3, function(a,b) {return Math.pow(a, b)}))
21
22 // hasil:
23 // 1,2, add: 3
24 // 43,21, subtract: 22
25 // 4^3: 64
```

Listing 3.1: Ekspresi Lambda di JavaScript

3.1.2. Higher-order Function

Higher-order function (sering disebut juga sebagai *functor* adalah suatu fungsi yang sedikit-tidaknya menggunakan satu atau lebih fungsi lain sebagai parameter dari fungsi, atau menghasilkan fungsi sebagai nilai kembalian.

```
1 function forEach(array, action) {
2     for (var i = 0; i < array.length; i++) {
3         action(array[i]);
4     }
5 }
6
7 function print(word) {
8     console.log(word);
9 }
10
11 function makeUpperCase(word) {
12     console.log(word.toUpperCase());
13 }
14
15 forEach(["satu", "dua", "tiga"], print);
16 forEach(["satu", "dua", "tiga"], makeUpperCase);
17
18 // hasil:
19 //satu
20 //dua
21 //tiga
22 //SATU
23 //DUA
```

```
23 //TIGA
```

Listing 3.2: Higher-order Function di JavaScript

3.1.3. Closure

Suatu *closure* merupakan definisi suatu fungsi bersama-sama dengan lingkungannya. Lingkungan tersebut terdiri atas fungsi internal serta berbagai variabel lokal yang masih tetap tersedia saat fungsi utama / closure tersebut selesai dieksekusi.

```
1 // Diambil dengan sedikit perubahan dari:
2 // https://developer.mozilla.org/en-US/docs/JavaScript/Guide/Closures
3 function makeAdder(x) {
4     return function(y) {
5         return x + y;
6     };
7 }
8
9 var add5 = makeAdder(5);
10 var add10 = makeAdder(10);
11
12 console.log(add5(2)); // 7
13 console.log(add10(2)); // 12
```

Listing 3.3: Closure di JavaScript

3.1.4. Currying

Currying memungkinkan pemrogram untuk membuat suatu fungsi dengan cara menggunakan fungsi yang sudah tersedia secara parsial, artinya tidak perlu menggunakan semua argumen dari fungsi yang sudah tersedia tersebut.

```
1 // Diambil dari:
2 // http://javascriptweblog.wordpress.com/2010/04/05/
3 //      curry-cooking-up-tastier-functions/
4 // dengan sedikit perubahan
5
6 function toArray(fromEnum) {
7     return Array.prototype.slice.call(fromEnum);
8 }
9
10 Function.prototype.curry = function() {
11     if (arguments.length < 1) {
12         return this; //nothing to curry with - return function
13     }
14     var __method = this;
15     var args = toArray(arguments);
16     return function() {
17         return __method.apply(this, args.concat(toArray(arguments)));
18     }
19 }
20
21 var add = function(a,b) {
22     return a + b;
23 }
24
25 //create function that returns 10 + argument
```

```
26 var addTen = add.curry(10);  
27 console.log(addTen(20)); //30
```

Listing 3.4: Currying di JavaScript

3.2. Pemrograman Berorientasi Obyek

3.2.1. Pengertian

Pemrograman Berorientasi Obyek (selanjutnya akan disingkat PBO) adalah suatu paradigma pemrograman yang memandang bahwa pemecahan masalah pemrograman akan dilakukan melalui definisi berbagai kelas kemudian membuat berbagai obyek berdasarkan kelas yang dibuat tersebut dan setelah itu mendefinisikan interaksi antar obyek tersebut dalam memecahkan masalah pemrograman. Obyek bisa saling berinteraksi karena setiap obyek mempunyai properti (sifat / karakteristik) dan *method* untuk mengerjakan suatu pekerjaan tertentu. Jadi, bisa dikatakan bahwa paradigma ini menggunakan cara pandang yang manusiawi dalam penyelesaian masalah.

Dengan demikian, inti dari PBO sebenarnya terletak pada kemampuan untuk mengabstraksikan berbagai obyek ke dalam kelas (yang terdiri atas properti serta method). Paradigma PBO biasanya juga mencakup *inheritance* atau pewarisan (sehingga terbentuk skema yang terdiri atas *superclass* dan *subclass*). Ciri lainnya adalah *polymorphism* dan *encapsulation* / pengkapsulan.

JavaScript adalah bahasa pemrograman yang mendukung PBO dan merupakan implementasi dari ECMAScript. Implementasi PBO di JavaScript adalah *prototype-based programming* yang merupakan salah satu subset dari PBO. Pada *prototype-based programming*, kelas / *class* tidak ada. Pewarisan diimplementasikan melalui *prototype*.

3.2.2. Definisi Obyek

Definisi obyek dilakukan dengan menggunakan definisi *function*, sementara *this* digunakan di dalam definisi untuk menunjukkan ke obyek tersebut. Sementara itu, `Kelas.prototype.namaMethod` digunakan untuk mendefinisikan method dengan nama `namaMethod` pada kelas `Kelas`. Perhatikan contoh pada listing berikut.

```
1 var url = require('url');  
2  
3 // Definisi obyek  
4 function Halaman(alamatUrl) {  
5     this.url = alamatUrl;  
6     console.log("Mengakses alamat " + alamatUrl);  
7 }  
8  
9 Halaman.prototype.getDomainName = function() {  
10     return url.parse(this.url, true).host;  
11 }  
12 // sampai disini definisi obyek  
13 // Halaman.prototype.getDomainName => menetapkan method getDomainName  
14 // untuk obyek  
15  
16 var halSatu = new Halaman("http://nodejs.org/api/http.html");
```

```

17 var halDua = new Halaman("http://bpdp.name/login?fromHome");
18
19 console.log("Alamat URL yang diakses oleh halSatu = " + halSatu.url);
20 console.log("Alamat URL yang diakses oleh halDua = " + halDua.url);
21
22 console.log("Nama domain halDua = " + halDua.getDomainName());
23
24 // hasil:
25 // Mengakses alamat http://nodejs.org/api/http.html
26 // Mengakses alamat http://bpdp.name/login?fromHome
27 // Alamat URL yang diakses oleh halSatu = http://nodejs.org/api/http.html
28 // Alamat URL yang diakses oleh halDua = http://bpdp.name/login?fromHome
29 // Nama domain halDua = bpdp.name

```

Listing 3.5: Definisi obyek di JavaScript

3.2.3. *Inheritance* / Pewarisan

Pewarisan di JavaScript bisa dicapai menggunakan *prototype*. Listing program berikut memperlihatkan bagaimana pewarisan diimplementasikan di JavaScript.

```

1 // Definisi obyek
2 function Kelas(param) {
3     this.property1 = new String(param);
4 }
5
6 Kelas.prototype.methodSatu = function() {
7     return this.property1;
8 }
9
10 var kelasSatu = new Kelas("ini parameter 1 dari kelas 1");
11
12 console.log("Property 1 dari kelasSatu = " + kelasSatu.property1);
13 console.log("Property 1 dari kelasSatu, diambil dari method = " + kelasSatu.methodSatu());
14
15 // Definisi inheritance:
16 // SubKelas merupakan anak dari Kelas yang didefinisikan
17 // di atas.
18
19 SubKelas.prototype = new Kelas();
20 SubKelas.prototype.constructor = SubKelas;
21
22 function SubKelas(param) {
23     this.property1 = new String(param);
24 }
25
26 // method overriding
27 SubKelas.prototype.methodSatu = function(keHurufBesar) {
28     console.log("Ubah ke huruf besar? = " + keHurufBesar);
29     if (keHurufBesar) {
30         return this.property1.toUpperCase();
31     } else {
32         return this.property1.toLowerCase();
33     }
34 }
35
36 SubKelas.prototype.methodDua = function() {
37     console.log("Berada di method dua dari SubKelas");
38 }
39

```

```
40 // mari diuji
41 var subKelasSatu = new SubKelas("Parameter 1 Dari Sub Kelas 1");
42
43 console.log("Property 1 dari sub kelas 1 = " + subKelasSatu.property1);
44 console.log("Property 1 dari sub kelas 1, dr method+param = " + subKelasSatu.methodSatu(true));
45 console.log("Property 1 dari sub kelas 1, dr method+param = " + subKelasSatu.methodSatu(false));
46
47 console.log(subKelasSatu.methodDua());
48 // hasil:
49 //
50 //Property 1 dari kelasSatu = ini parameter 1 dari kelas 1
51 //Property 1 dari kelasSatu, diambil dari method = ini
52 //parameter 1 dari kelas 1
53 //Property 1 dari sub kelas 1 = Parameter 1 Dari Sub Kelas 1
54 //Ubah ke huruf besar? = true
55 //Property 1 dari sub kelas 1, dr method+param =
56 //PARAMETER 1 DARI SUB KELAS 1
57 //Ubah ke huruf besar? = false
58 //Property 1 dari sub kelas 1, dr method+param =
59 //parameter 1 dari sub kelas 1
60 //Berada di method dua dari SubKelas
```

Listing 3.6: Pewarisan di PBO JavaScript

Mengelola Paket Menggunakan npm

4.1. Apakah npm Itu?

Node.js memungkinkan developer untuk mengembangkan aplikasi secara modular dengan memisahkan berbagai komponen *reusable code* ke dalam pustaka (*library*). Berbagai pustaka tersebut bisa diperoleh di <http://npmjs.org>. Node.js menyediakan perintah *npm* untuk mengelola paket pustaka di repositori tersebut. Untuk menggunakan utilitas ini, pemrogram harus terkoneksi dengan Internet.

4.2. Menggunakan npm

Saat melakukan instalasi Node.js, secara otomatis *npm* akan disertakan. Dengan perintah *npm* tersebut, seorang pemrogram bisa mengelola pustaka yang tersedia di repositori. Jika pemrogram mempunyai pustakan yang bisa digunakan oleh orang lain, maka pemrogram yang bersangkutan juga bisa menyimpan pustaka tersebut ke dalam repositori sehingga memungkinkan untuk diinstall oleh pemrogram-pemrogram lain di seluruh dunia. Sintaksis lengkap dari penggunaan perintah *npm* ini adalah sebagai berikut¹:

```
1 $ npm --help
2
3 Usage: npm <command>
4
5 where <command> is one of:
6   add-user, adduser, apihelp, author, bin, bugs, c, cache,
7   completion, config, ddp, dedupe, deprecate, docs, edit,
8   explore, faq, find, find-dupes, get, help, help-search,
9   home, i, info, init, install, isntall, issues, la, link,
10  list, ll, ln, login, ls, outdated, owner, pack, prefix,
11  prune, publish, r, rb, rebuild, remove, restart, rm, root,
12  run-script, s, se, search, set, show, shrinkwrap, star,
```

¹beberapa bagian tertulis spesifik lokasi direktori di komputer yang digunakan penulis

```

13 stars, start, stop, submodule, tag, test, tst, un,
14 uninstall, unlink, unpublish, unstar, up, update, version,
15 view, whoami
16
17 npm <cmd> -h      quick help on <cmd>
18 npm -l            display full usage info
19 npm faq           commonly asked questions
20 npm help <term>   search for help on <term>
21 npm help npm      involved overview
22
23 Specify configs in the ini-formatted file:
24   /home/bpdp/.npmrc
25 or on the command line via: npm <command> --key value
26 Config info can be viewed via: npm help config
27
28 npm@1.2.18 /home/bpdp/software/node-v0.10.5-linux-x86/lib/node_modules/npm

```

Listing 4.1: Sintaksis lengkap perintah *npm*

Pada bagian berikut, kita akan membahas lebih lanjut penggunaan perintah *npm* tersebut.

4.2.1. Instalasi Paket

npm sebenarnya bukan merupakan singkatan dari *Node Package Manager*, meskipun seringkali orang menerjemahkan dengan singkatan tersebut dan npm seharusnya ditulis dalam huruf kecil semua seperti yang dijelaskan pada FAQ (*Frequently Asked Questions*)². npm merupakan bilah alat berbasis baris perintah, dijalankan melalui shell atau *command prompt*. Sama seperti kebanyakan bilah alat berbasis baris perintah lain, npm memiliki struktur perintah *npm perintah argumen*. Instalasi paket pustaka dilakukan dengan perintah berikut :

```

1 $ npm install namapaket

```

Listing 4.2: Cara install paket menggunakan npm

Perintah diatas akan memasang versi terakhir dari paket “namapaket”. Selain itu *npm* juga dapat memasang paket langsung pada sebuah folder, tarball atau tautan untuk sebuah tarball.

4.2.2. Struktur Instalasi Paket Node.js

Dalam instalasi paket pustaka, berkas-berkas akan terletak dalam folder lokal aplikasi *node_modules*. Pada mode instalasi paket pustaka global (dengan -g atau --global dibelakang baris perintah), paket pustaka akan dipasang pada */usr/lib/node_modules* (dengan lokasi instalasi Node.js standar). Mode global memungkinkan paket pustaka digunakan tanpa memasang paket pustaka pada setiap folder lokal aplikasi. Mode global ini juga membutuhkan hak administrasi lebih (sudo atau root) dari pengguna agar dapat menulis pada lokasi standar.

²<https://npmjs.org/doc/faq.html>

Jika berada pada direktori `$HOME`, maka paket-paket npm tersebut akan terinstall di `$HOME/.npm`, sedangkan jika kita berada di luar direktori `$HOME`, maka paket-paket tersebut akan terinstall di `$CWD/node_modules` (`$CWD = Current Working Directory` - direktori aktif saat ini). Daftar paket pustaka yang terpasang dapat dilihat menggunakan perintah berikut:

```
1 $ npm ls
2 —> untuk melihat pada $CWD
3     atau
4 $ npm ls -g
5 —> untuk melihat pada direktori global
```

Listing 4.3: Argumen npm untuk melihat daftar paket terpasang

Selain melihat daftar paket pustaka yang digunakan dalam aplikasi maupun global, perintah diatas juga akan menampilkan paket dependensi dalam struktur pohon. Jika kita belum menginstall paket-paket yang diperlukan, akan muncul peringatan. Berikut ini adalah contoh peringatan dari paket-paket yang belum terinstall di aplikasi hello-express saat mengerjakan perintah “npm ls” di direktori tempat aplikasi tersebut berada (lihat bab 1):

```
1 $ npm ls
2 npm WARN package.json hello@0.0.1 No README.md file found!
3 hello@0.0.1 /home/bdp/kerjaan/git-repos/buku-cloud-nodejs/src/bab-01/hello
4 +-- UNMET DEPENDENCY express 3.2.2
5 +-- UNMET DEPENDENCY jade *
6
7 npm ERR! missing: express@3.2.2, required by hello@0.0.1
8 npm ERR! missing: jade@*, required by hello@0.0.1
9 npm ERR! not ok code 0
10 $
```

Listing 4.4: npm ls pada aplikasi yang paket-paketnya belum terinstall

Jika sudah terinstall, perintah “npm ls” akan menampilkan struktur dari paket yang telah terinstall dalam bentuk struktur pohon seperti pada Gambar 4.1.

4.2.3. Menghapus Paket / *Uninstall*

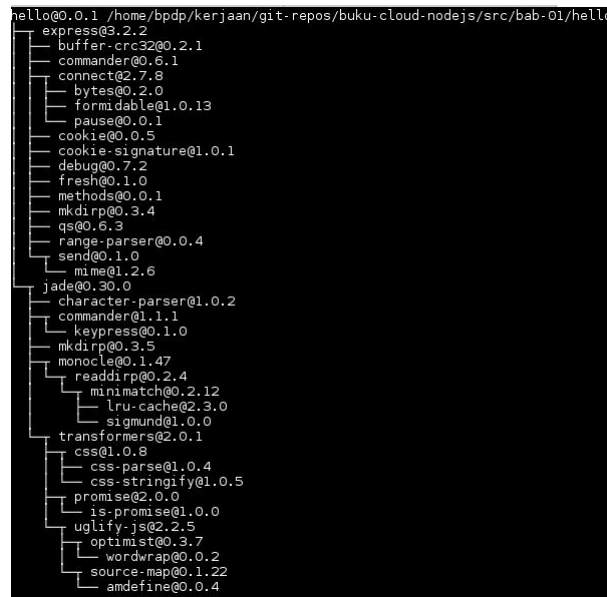
Menghapus paket pustaka menggunakan npm pada dasarnya hampir sama dengan saat memasang paket, namun dengan perintah *uninstall*. Berikut perintah lengkapnya.

```
1 $ npm uninstall namapaket
2 —> uninstall namapaket di $CWD/node_modules
3     atau
4 $ npm uninstall namapaket -g
5 —> uninstall paket di dir global
```

Listing 4.5: Perintah menghapus paket di npm

4.2.4. Mencari Paket

Untuk mencari paket, gunakan argumen *search* dan nama atau bagian dari nama paket yang dicari. Contoh berikut ini akan mencari paket dengan kata kunci ‘sha512’ (tampilan berikut merupakan tampilan yang terpotong):



Gambar 4.1.: Tampilan “npm ls” pada direktori proyek dengan paket terinstall lengkap

```

1 $ npm search sha512
2 npm http GET https://registry.npmjs.org/-/all/s...
3 npm http 200 https://registry.npmjs.org/-/all/s...
4 NAME      DESCRIPTION      ...
5 jshashes  A fast and independent hashing librar...
6 krypto    High-level crypto library, making the...
7 passhash  Easily and securely hash passwords wi...
8 pwhash    Generate password hashes from the com...

```

Listing 4.6: Perintah menghapus paket di npm

Setelah menemukan paketnya, pemrogram bisa menginstall langsung ataupun melihat informasi lebih lanjut tentang pustakan tersebut.

4.2.5. Menampilkan Informasi Paket

Setelah mengetahui nama paket, pemrogram bisa memperoleh informasi lebih lanjut dalam format JSON menggunakan parameter *view*. Contoh dibawah ini menampilkan rincian dalam format JSON dari paket *arango.client*:

```

1 $ npm view arango.client
2 npm http GET https://registry.npmjs.org/arango.client
3 npm http 200 https://registry.npmjs.org/arango.client
4
5 { name: 'arango.client',
6   description: 'ArangoDB javascript client',
7   'dist-tags': { latest: '0.5.6' },
8   versions:
9     [ '0.3.1',

```

```

10   '0.3.2',
11   '0.4.0',
12   '0.5.0',
13   '0.5.1',
14   '0.5.4',
15   '0.5.6' ],
16   maintainers: 'kaerus <anders@kaerus.com>',
17   time:
18     { '0.3.1': '2012-08-09T12:04:34.594Z',
19       '0.3.2': '2012-08-09T12:49:02.322Z',
20       '0.4.0': '2012-09-17T10:44:43.187Z',
21       '0.5.0': '2012-10-01T14:51:32.668Z',
22       '0.5.1': '2012-10-03T22:11:58.376Z',
23       '0.5.4': '2012-10-16T09:45:37.477Z',
24       '0.5.6': '2012-10-26T17:34:28.491Z' },
25   author: 'Kaerus <contact@kaerus.com> (http://kaerus.com)',
26   repository:
27     { type: 'git',
28       url: 'git://github.com/kaerus/arango-client.git' },
29   version: '0.5.6',
30   keywords:
31     [ 'arango',
32       'nosql',
33       'qunit',
34       'amd' ],
35   contributors: 'anders elo <anders@kaerus.com>',
36   dependencies: { amdefine: '>=0.0.2' },
37   devDependencies: { requirejs: '>=2.0.6' },
38   bugs: { url: 'https://github.com/kaerus/arango-client/issues' },
39   main: 'index.js',
40   license: 'MIT',
41   dist:
42     { shasum: '48279e7cf9ea0b4b6766f09671224c46d6e716b0',
43       tarball: 'http://registry.npmjs.org/arango-client/-/arango-client-0.5.6.tgz' },
44   directories: {} }
45 $

```

Listing 4.7: Menampilkan rincian suatu paket dalam format JSON

4.2.6. Memperbaharui Paket

Jika terdapat versi baru, kita bisa memperbaharui secara otomatis menggunakan argumen *update* berikut ini:

```

1 $ npm update
2 —> update paket di $PWD/node_modules
3 $ npm update -g
4 —> update paket global

```

Listing 4.8: Memperbaharui paket

Node.js dan Web: Teknik Pengembangan Aplikasi

5.1. Pendahuluan

Pada saat membangun aplikasi Cloud dengan antarmuka web menggunakan Node.js, ada beberapa teknik pemrograman yang bisa digunakan. Bab ini akan membahas berbagai teknik tersebut. Untuk mengerjakan beberapa latihan di bab ini, digunakan suatu file dengan format JSON. File *pegawai.json* berikut ini akan digunakan dalam pembahasan selanjutnya.

```
1 {  
2   "pegawai": [  
3     {  
4       "id": "1",  
5       "nama": "Zaky",  
6       "alamat": "Purwomartani"  
7     },  
8     {  
9       "id": "2",  
10      "nama": "Ahmad",  
11      "alamat": "Kalasan"  
12    },  
13    {  
14      "id": "3",  
15      "name": "Aditya",  
16      "alamat": "Sleman"  
17    }  
18  ]  
19 }
```

Listing 5.1: pegawai.json

Jika ingin memeriksa validitas dari data berformat JSON, pemrogram bisa menggunakan validator di <http://jsonlint.com>.

5.2. Event-Driven Programming dan EventEmitter

Event-Driven Programming (selanjutnya akan disebut EDP) atau sering juga disebut *Event-Based Programming* merupakan teknik pemrograman yang menggunakan *event* atau suatu kejadian tertentu sebagai pemicu munculnya suatu aksi serta aliran program. Contoh event misalnya adalah sebagai berikut:

- ✓ Menu dipilih.
- ✓ Tombol "Submit" di klik.
- ✓ Server menerima permintaan dari klien.

Pada dasarnya ada beberapa bagian yang harus disiapkan dari paradigma dan teknik pemrograman ini:

- ✓ *main loop* atau suatu konstruksi utama program yang menunggu dan mengirimkan sinyal event.
- ✓ definisi dari berbagai event yang mungkin muncul
- ✓ definisi *event-handler* untuk menangani event yang muncul dan dikirimkan oleh *main loop*

Node.js merupakan peranti pengembangan yang menggunakan teknik pemrograman ini. Pada Node.js, EDP ini semua dikendalikan oleh kelas *events.EventEmitter*. Jika ingin menggunakan kelas ini, gunakan *require('events')*. Dalam terminologi Node.js, jika suatu event terjadi, maka dikatakan sebagai *emits an event*, sehingga kelas yang digunakan untuk menangani itu disebut dengan *events.EventEmitter*. Pada dasarnya banyak event yang digunakan oleh berbagai kelas lain di Node.js. Contoh kecil dari penggunaan itu diantaranya adalah *net.Server* yang meng-*emit* event "connection", "listening", "close", dan "error".

Untuk memahami mekanisme ini, pahami dua kode sumber berikut:

- ✓ *server.js*: mengaktifkan server http (diambil dari manual Node.js)
- ✓ *server-on-error.js*: mencoba mengaktifkan server pada host dan port yang sama dengan *server.js*. Aktivasi ini akan menyebabkan Node.js meng-*emit* event 'error' karena host dan port sudah digunakan di *server.js*.

File *server.js* dijalankan lebih dulu, setelah itu baru menjalankan *server-on-error.js*.

```
1 var http = require('http');
2
3 http.createServer(function (req, res) {
4
5     res.writeHead(200, {'Content-Type': 'text/plain'});
6     res.end('Hello World\n');
7
8 }).listen(1337, '127.0.0.1');
9
10 console.log('Server running at http://127.0.0.1:1337/');
```

Listing 5.2: server.js

```
1 var net = require('net');
2
3 var server = net.createServer(function(sock) {
4
5     // Event dan event-handler
6     // 'data' => jika ada data yang dikirimkan dari klien
7     sock.on('data', function(data) {
8         console.log('data ' + sock.remoteAddress + ': ' + data);
9     });
10
11     // 'close' => jika koneksi ditutup
12     sock.on('close', function(data) {
13         console.log('koneksi ditutup');
14     });
15
16 });
17
18 server.listen(1337, function() {
19     console.log('Server aktif di 127.0.0.1:1337');
20 });
21
22 server.on('error', function (e) {
23
24     if (e.code === 'EADDRINUSE') {
25         console.log('Error: host dan port sudah digunakan.');
```

Listing 5.3: server-on-error.js

5.3. Asynchronous / Non-blocking IO dan Callback

Asynchronous input/output merupakan suatu bentuk pemrosesan masukan/keluaran yang memungkinkan pemrosesan dilanjutkan tanpa menunggu proses tersebut selesai. Saat pemrosesan masukan/keluaran tersebut selesai, hasil akan diberikan ke suatu fungsi. Fungsi yang menangani hasil pemrosesan saat pemrosesan tersebut selesai disebut *callback* (pemanggilan kembali). Jadi, mekanismenya adalah: proses masukan/keluaran - lanjut ke alur berikutnya - panggil kembali fungsi pemroses jika proses masukan/keluaran sudah selesai.

```
1 var fs = require('fs');
2 var sys = require('sys');
3
4 sys.puts('Mulai baca file');
5 data = fs.readFileSync('./pegawai.json', "utf-8");
6 console.log(data);
7 sys.puts('Baris setelah membaca file');
8
9 // hasil:
10 //Mulai baca file
11 //{
12 //  "pegawai": [
13 //    {
14 //      "id": "1",
15 //      "nama": "Zaky",
```

```
16 //      "alamat": "Purwomartani"
17 //    },
18 //    {
19 //      "id": "2",
20 //      "nama": "Ahmad",
21 //      "alamat": "Kalasan"
22 //    },
23 //    {
24 //      "id": "3",
25 //      "name": "Aditya",
26 //      "alamat": "Sleman"
27 //    }
28 //  ]
29 //}
30 //
31 //Baris setelah membaca file
```

Listing 5.4: Membaca file secara synchronous

```
1 var fs = require('fs');
2 var sys = require('sys');
3
4 sys.puts('Mulai baca file');
5 fs.readFile('./pegawai.json', "utf-8", function(err, data) {
6   if (err) throw err;
7   console.log(data);
8 })
9 sys.puts('Baris setelah membaca file');
10
11 // hasil:
12 //Mulai baca file
13 //Baris setelah membaca file
14 //{
15 //  "pegawai": [
16 //    {
17 //      "id": "1",
18 //      "nama": "Zaky",
19 //      "alamat": "Purwomartani"
20 //    },
21 //    {
22 //      "id": "2",
23 //      "nama": "Ahmad",
24 //      "alamat": "Kalasan"
25 //    },
26 //    {
27 //      "id": "3",
28 //      "name": "Aditya",
29 //      "alamat": "Sleman"
30 //    }
31 //  ]
32 //}
```

Listing 5.5: Membaca file secara asynchronous

Mengakses Basis Data NoSQL: mongoDB

6.1. Apa itu Basis Data NoSQL?

Pada awalnya, istilah NoSQL digunakan oleh Carlo Strozzi untuk menyebut nama software basis data yang dibuat olehnya. Software basis data tersebut tidak mengikuti standar SQL, sehingga dia menyebut software tersebut dengan "NoSQL"¹. Setelah itu, istilah NoSQL dipopulerkan oleh Eric Evans untuk menyebut jenis software basis data yang tidak menggunakan standar SQL. Dalam perkembangan berikutnya, NoSQL ini lebih diarahkan pada "Not Only SQL" dan digunakan untuk kategorisasi basis data *non-relational* (misalnya OODBMS, Graph Database, Document-oriented, dan lain-lain). Meski ada usaha untuk menstandarkan bahasa *query* untuk NoSQL (UnQL - *Unstructured Query Language*), sampai saat ini usaha tersebut tidak menghasilkan sesuatu hal yang disepakati bersama karena dunia NoSQL memang kompleks sekali. Untuk melihat daftar dari basis data NoSQL, anda bisa melihat ke <http://nosql-databases.org>.

6.2. Mengenal mongoDB dan Fitur-fiturnya

mongoDB adalah salah satu software NoSQL yang termasuk dalam kategori *Document Store / Document-Oriented Database*, yaitu data disimpan dalam bentuk dokumen. Suatu dokumen bisa diibaratkan seperti suatu *record* dalam basis data relasional dan isi dari masing-masing dokumen tersebut bisa berbeda-beda dan ada pula yang sama. Hal ini berbeda dengan basis data relasional yang menetapkan keseragaman kolom serta tipe data dengan data yang NULL jika tidak terdapat data. mongoDB menyimpan data dalam bentuk dokumen dengan menggunakan format JSON. Berikut adalah fitur dari mongoDB:

¹http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/nosql/Home%20Page

- ✓ menggunakan format JSON dalam penyimpanan data
- ✓ mendukung indeks
- ✓ mendukung replikasi
- ✓ auto-sharding untuk skalabilitas horizontal
- ✓ query yang lengkap
- ✓ pembaruan data yang cepat
- ✓ mendukung Map/Reduce
- ✓ mendukung GridFS

6.2.1. Memulai Server

Seperti halnya basis data relasional seperti MySQL, PostgreSQL, dan lain-lain, mongoDB juga memulai dengan menjalankan server yang memungkinkan server tersebut melayani permintaan akses data dokumen melalui klien. Untuk memulai server, siapkan direktori yang akan menjadi tempat menyimpan data (defaultnya adalah /data/db). Jika menginginkan lokasi lain, gunakan argumen `-dbpath` saat menjalankan server sebagai berikut (buat direktorinya jika belum ada):

```

1 $ mkdir data
2 $ mongod --rest --dbpath ./data
3 Sun May 5 22:47:05.495
4 Sun May 5 22:47:05.495 warning: 32-bit servers don't have journaling
5   enabled by default. Please use --journal if you want durability.
6 Sun May 5 22:47:05.495
7 Sun May 5 22:47:05.581 [initandlisten] MongoDB starting : pid=2994
8   port=27017 dbpath=./data 32-bit host=bdp-arch
9 Sun May 5 22:47:05.581 [initandlisten]
10 Sun May 5 22:47:05.581 [initandlisten] ** NOTE: This is a 32 bit
11   MongoDB binary.
12 Sun May 5 22:47:05.581 [initandlisten] **      32 bit builds are
13   limited to less than 2GB of data (or less with --journal).
14 Sun May 5 22:47:05.581 [initandlisten] **      Note that journaling
15   defaults to off for 32 bit and is currently off.
16 Sun May 5 22:47:05.581 [initandlisten] **      See
17   http://dochub.mongodb.org/core/32bit
18 Sun May 5 22:47:05.581 [initandlisten]
19 Sun May 5 22:47:05.581 [initandlisten] db version v2.4.4-pre-
20 Sun May 5 22:47:05.581 [initandlisten] git version: nogitversion
21 Sun May 5 22:47:05.581 [initandlisten] build info: Linux fyan
22   3.8.3-2-ARCH #1 SMP PREEMPT Sun Mar 17 13:04:22 CET 2013
23   i686 BOOST_LIB_VERSION=1_53
24 Sun May 5 22:47:05.581 [initandlisten] allocator: system
25 Sun May 5 22:47:05.581 [initandlisten] options: { dbpath:
26   "./data", rest: true }
27 Sun May 5 22:47:05.789 [FileAllocator] allocating new datafile
28   ./data/local.ns, filling with zeroes...
29 Sun May 5 22:47:05.790 [FileAllocator] creating directory
30   ./data/_tmp
31 Sun May 5 22:47:05.997 [FileAllocator] done allocating datafile
32   ./data/local.ns, size: 16MB, took 0.088 secs

```



```

33 Sun May 5 22:47:05.998 [FileAllocator] allocating new datafile
34 ./data/local.0, filling with zeroes...
35 Sun May 5 22:47:06.086 [FileAllocator] done allocating datafile
36 ./data/local.0, size: 16MB, took 0.088 secs
37 Sun May 5 22:47:06.091 [initandlisten] command local.$cmd command:
38 { create: "startup_log", size: 10485760, capped: true }
39 ntoreturn:1 keyUpdates:0 reslen:37 301ms
40 Sun May 5 22:47:06.093 [initandlisten] waiting for connections
41 on port 27017
42 Sun May 5 22:47:06.093 [websvr] admin web console waiting for
43 connections on port 28017

```

Listing 6.1: Menjalankan server MongoDB (mongod)

Untuk mengakhiri server, tekan *Ctrl-C*, mongoDB akan mengakhiri server sebagai berikut:

```

1 ^CSun May 5 22:52:52.344 [signalProcessingThread] got
2 signal 2 (Interrupt), will terminate after current cmd ends
3 Sun May 5 22:52:52.345 [signalProcessingThread] now exiting
4 Sun May 5 22:52:52.345 dbexit:
5 Sun May 5 22:52:52.345 [signalProcessingThread] shutdown:
6 going to close listening sockets...
7 Sun May 5 22:52:52.345 [signalProcessingThread] closing
8 listening socket: 7
9 Sun May 5 22:52:52.345 [signalProcessingThread] closing
10 listening socket: 8
11 Sun May 5 22:52:52.345 [signalProcessingThread] closing
12 listening socket: 9
13 Sun May 5 22:52:52.345 [signalProcessingThread] removing
14 socket file: /tmp/mongodb-27017.sock
15 Sun May 5 22:52:52.345 [signalProcessingThread] shutdown:
16 going to flush diaglog...
17 Sun May 5 22:52:52.345 [signalProcessingThread] shutdown:
18 going to close sockets...
19 Sun May 5 22:52:52.345 [signalProcessingThread] shutdown:
20 waiting for fs preallocator...
21 Sun May 5 22:52:52.345 [signalProcessingThread] shutdown:
22 closing all files...
23 Sun May 5 22:52:52.345 [signalProcessingThread] closeAllFiles()
24 finished
25 Sun May 5 22:52:52.346 [signalProcessingThread] shutdown:
26 removing fs lock...
27 Sun May 5 22:52:52.346 dbexit: really exiting now

```

Listing 6.2: Mengakhiri server MongoDB (mongod)

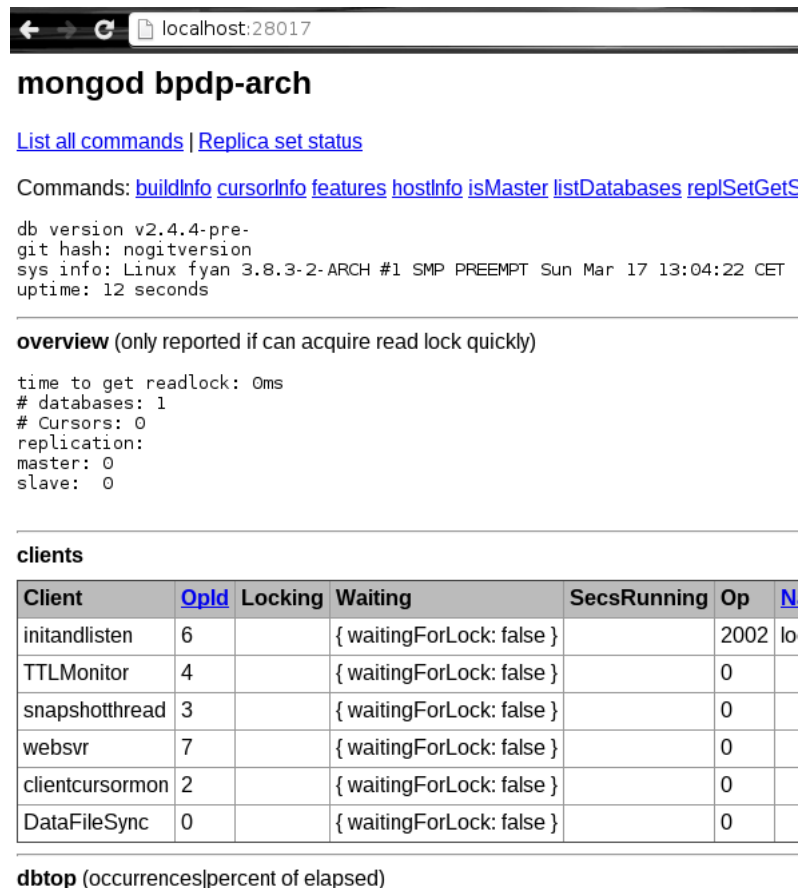
6.2.2. Klien dan Shell mongoDB

Setelah server hidup, pemrogram bisa menggunakan antarmuka administrasi web maupun menggunakan shell. *Admin web console* bisa diakses menggunakan port 28017 seperti pada gambar 6.1. Sementara itu, untuk mengakses server menggunakan shell, bisa digunakan perintah *mongo* sebagai berikut:

```

1 mongo
2 MongoDB shell version: 2.4.4-pre-
3 connecting to: test
4 Server has startup warnings:

```



← → ↻ localhost:28017

mongod bdpd-arch

[List all commands](#) | [Replica set status](#)

Commands: [buildInfo](#) [cursorInfo](#) [features](#) [hostInfo](#) [isMaster](#) [listDatabases](#) [replSetGetS](#)

db version v2.4.4-pre-
git hash: nogitversion
sys info: Linux fyan 3.8.3-2-ARCH #1 SMP PREEMPT Sun Mar 17 13:04:22 CET
uptime: 12 seconds

overview (only reported if can acquire read lock quickly)

time to get readlock: 0ms
databases: 1
Cursors: 0
replication:
master: 0
slave: 0

clients

Client	OpId	Locking	Waiting	SecsRunning	Op	N
initandlisten	6		{ waitingForLock: false }		2002	lo
TTLMonitor	4		{ waitingForLock: false }		0	
snapshotthread	3		{ waitingForLock: false }		0	
websvr	7		{ waitingForLock: false }		0	
clientcursormon	2		{ waitingForLock: false }		0	
DataFileSync	0		{ waitingForLock: false }		0	

dbtop (occurrences|percent of elapsed)

Gambar 6.1.: Admin web console untuk mongoDB

```

5 Sun May 5 22:56:00.015 [initandlisten]
6 Sun May 5 22:56:00.015 [initandlisten] ** NOTE: This is
7   a 32 bit MongoDB binary.
8 Sun May 5 22:56:00.015 [initandlisten] **           32 bit
9   builds are limited to less than 2GB of data (or less
10  with --journal).
11 Sun May 5 22:56:00.015 [initandlisten] **           Note that
12   journaling defaults to off for 32 bit and is currently off.
13 Sun May 5 22:56:00.015 [initandlisten] **           See
14   http://dochub.mongodb.org/core/32bit
15 Sun May 5 22:56:00.015 [initandlisten]
16 >

```

Listing 6.3: Shell mongoDB (mongo)

6.2.3. Documents dan Collections

Konsep dasar yang harus dipahami dalam mongoDB sebagai *document-oriented database* adalah *documents* dan *collections*. Sama halnya dengan basis data relasional, mongoDB menyimpan data dalam suatu basis data. Di dalam basis data tersebut terdapat *collections* yang bisa diibaratkan seperti tabel dalam basis data relasional. *Collections* digunakan untuk menyimpan dokumen (*documents*). Dalam istilah basis data relasional, *documents* adalah *records*. Kerjakan latihan berikut untuk memahami pengertian dari *documents* dan *collections*.

```

1  $ mongo
2  MongoDB shell version: 2.4.4-pre
3  connecting to: test
4  ...
5  ... [ jgn pedulikan warning ini ]
6  ...
7  ...
8  > db
9  test
10 > use mydb
11 switched to db mydb
12 > show dbs
13 local (empty)
14 > emp1 = { name : "Zaky", address : "Griya Purwa Asri" }
15 { "name" : "Zaky", "address" : "Griya Purwa Asri" }
16 > emp2 = { name : "Ahmad", address : "Purwomartani", email : "zakyahmadaditya@gmail.com" }
17 {
18     "name" : "Ahmad",
19     "address" : "Purwomartani",
20     "email" : "zakyahmadaditya@gmail.com"
21 }
22 > emp3 = { name : "Aditya", address : "Kalasan", phone: "08787878787" }
23 { "name" : "Aditya", "address" : "Kalasan", "phone" : "08787878787" }
24 > db.employees.insert( emp1 )
25 > db.employees.insert( emp2 )
26 > db.employees.insert( emp3 )
27 > show dbs
28 local (empty)
29 mydb 0.0625GB
30 > db
31 mydb
32 > show collections
33 employees
34 system.indexes
35 > db.employees.find()
36 { "_id" : ObjectId("50c74b63a7f83cba11e6b21e"), "name" : "Zaky", "address" :
37     "Griya Purwa Asri" }
38 { "_id" : ObjectId("50c74b6da7f83cba11e6b21f"), "name" : "Ahmad", "address" :
39     "Purwomartani", "email" : "zakyahmadaditya@gmail.com" }
40 { "_id" : ObjectId("50c74b79a7f83cba11e6b220"), "name" : "Aditya", "address" :
41     "Kalasan", "phone" : "08787878787" }
42 > db.employees.find( {name : "Ahmad"} )
43 { "_id" : ObjectId("50c74b6da7f83cba11e6b21f"), "name" : "Ahmad", "address" :
44     "Purwomartani", "email" : "zakyahmadaditya@gmail.com" }
45 > db.employees.findOne()
46 {
47     "_id" : ObjectId("50c74b63a7f83cba11e6b21e"),
48     "name" : "Zaky",
49     "address" : "Griya Purwa Asri"
50 }

```

```

51 > db.employees.find().limit(2)
52 { "_id" : ObjectId("50c74b63a7f83cba11e6b21e"), "name" : "Zaky", "address" :
53   "Griya Purwa Asri" }
54 { "_id" : ObjectId("50c74b6da7f83cba11e6b21f"), "name" : "Ahmad", "address" :
55   "Purwomartani", "email" : "zakyahmadaditya@gmail.com" }
56 >

```

Listing 6.4: Sesi dalam shell mongoDB

Basis data mongoDB hanya akan dibuat jika sudah dilakukan perintah untuk menyisipkan atau mengisi data *documents* ke dalam *collections* seperti perintah di atas.

6.3. Node.js dan MongoDB

6.3.1. Node-gyp

Node-gyp merupakan *native add-on build tool*, berfungsi untuk membantu proses kompilasi modul add-on native di Node.js. Node-gyp merupakan software bebas dan bisa diinstall menggunakan npm:

```
1 $ npm install -g node-gyp
```

Listing 6.5: Instalasi node-gyp

Node-gyp ini diinstall pada lokasi global. Pada materi ini, Node-gyp diperlukan untuk membangun *driver* dari mongoDB sehingga mongoDB bisa diakses oleh Node.js.

6.3.2. Driver Node.js untuk mongoDB

Mengakses mongoDB dari Node.js bisa dilakukan dengan menggunakan driver atau berbagai *wrapper* serta solusi sejenis ORM *Object-Relational Mapping*. Salah satu solusi yang tersedia adalah paket **mongodb**.

```

1 $ npm install mongodb
2 npm http GET https://registry.npmjs.org/mongodb
3 npm http 200 https://registry.npmjs.org/mongodb
4 npm http GET https://registry.npmjs.org/mongodb/-/mongodb-1.3.0.tgz
5 npm http 200 https://registry.npmjs.org/mongodb/-/mongodb-1.3.0.tgz
6 npm http GET https://registry.npmjs.org/kerberos
7 npm http GET https://registry.npmjs.org/bson/0.1.8
8 npm http 200 https://registry.npmjs.org/kerberos
9 npm http GET https://registry.npmjs.org/kerberos/-/kerberos-0.0.2.tgz
10 npm http 200 https://registry.npmjs.org/bson/0.1.8
11 npm http GET https://registry.npmjs.org/bson/-/bson-0.1.8.tgz
12 npm http 200 https://registry.npmjs.org/kerberos/-/kerberos-0.0.2.tgz
13 npm http 200 https://registry.npmjs.org/bson/-/bson-0.1.8.tgz
14
15 > kerberos@0.0.2 install /home/bpdp/kerjaan/git-repos/buku-cloud-nodejs/
16   src/bab-06/node_modules/mongodb/node_modules/kerberos
17 > (node-gyp rebuild 2> builderror.log) || (exit 0)
18
19
20 > bson@0.1.8 install /home/bpdp/kerjaan/git-repos/buku-cloud-nodejs/src/
21   bab-06/node_modules/mongodb/node_modules/bson
22 > (node-gyp rebuild 2> builderror.log) || (exit 0)
23

```

```

24 mongodb@1.3.0 node_modules/mongodb
25 +--- bson@0.1.8
26 +--- kerberos@0.0.2
27 $

```

Listing 6.6: Instalasi driver mongoDB

Solusi lain yang bisa digunakan antara lain adalah:

- ✓ Mongoose (<http://mongoosejs.com/>)
- ✓ Mongojs (<https://github.com/gett/mongojs>)
- ✓ Mongolia (<https://github.com/masylum/mongolia>)
- ✓ Mongoskin (<https://github.com/kissjs/node-mongoskin>)

6.3.3. Mengakses mongoDB dari Node.js

Dengan menggunakan *collections* dan *documents* di atas, kita akan mengakses data tersebut menggunakan Node.js. Untuk lebih menyederhanakan, kita akan menggunakan *wrapper* dari mongoDB native driver, yaitu Mongojs. Install Mongojs lebih dahulu menggunakan npm:

```

1 $ npm install mongojs
2 npm http GET https://registry.npmjs.org/mongojs
3 npm http 200 https://registry.npmjs.org/mongojs
4 npm http GET https://registry.npmjs.org/mongojs/-/mongojs-0.7.4.tgz
5 npm http 200 https://registry.npmjs.org/mongojs/-/mongojs-0.7.4.tgz
6 npm http GET https://registry.npmjs.org/readable-stream
7 npm http GET https://registry.npmjs.org/thunky
8 npm http 200 https://registry.npmjs.org/readable-stream
9 npm http 200 https://registry.npmjs.org/thunky
10 mongojs@0.7.4 node_modules/mongojs
11 +--- thunky@0.1.0
12 +--- readable-stream@1.0.2
13 $

```

Listing 6.7: Instalasi wrapper mongojs

Setelah itu, buat program sesuai dengan listing program berikut.

```

1 var databaseUrl = "localhost/mydb";
2 var collections = ["employees"];
3 var db = require("mongojs").connect(databaseUrl, collections);
4
5 // mencari pegawai bernama Aditya
6 db.employees.find({name: "Aditya"}, function(err, employees) {
7   if( err || !employees) console.log("Tidak ada pegawai dengan nama Aditya");
8   else employees.forEach( function(emps) {
9     console.log(emps);
10   });
11 });
12
13 // menyimpan data pegawai baru: Bambang
14 db.employees.save({name : "Bambang", address : "Yogyakarta", password: "ealhadalah",
15   sex: "male"}, function(err, saved) {
16   if( err || !saved ) console.log("Pegawai 'Bambang' gagal disimpan");

```

```

17     else console.log("Data pegawai 'Bambang' tersimpan");
18   });
19
20   // mengupdate data pegawai: Ahmad
21   db.employees.update({name : "Ahmad"}, {$set: {address: "Finlandia"}},
22     function(err, updated) {
23       if( err || !updated ) console.log("Data 'Ahmad' gagal diperbaharui");
24       else console.log("Data 'Ahmad' berhasil diperbaharui");
25     });
26
27   // Hasil:
28   //{ _id: 50c74b79a7f83cba11e6b220 ,
29   //  name: 'Aditya',
30   //  address: 'Kalasan',
31   //  phone: '08787878787' }
32   //Data pegawai 'Bambang' tersimpan
33   //Data 'Ahmad' berhasil diperbaharui
34   //
35   // Hasil di db:
36   //> db.employees.find()
37   //{ "_id" : ObjectId("50c74b63a7f83cba11e6b21e"), "name" :
38   //  "Zaky", "address" : "Griya Purwa Asri" }
39   //{ "_id" : ObjectId("50c74b6da7f83cba11e6b21f"), "address" :
40   //  "Finlandia", "email" : "zakyahmadaditya@gmail.com", "name" : "Ahmad" }
41   //{ "_id" : ObjectId("50c74b79a7f83cba11e6b220"), "name" :
42   //  "Aditya", "address" : "Kalasan", "phone" : "08787878787" }
43   //{ "name" : "Bambang", "address" : "Yogyakarta", "password" :
44   //  "ealhadalah", "sex" : "male", "_id" :
45   //  ObjectId("50c75d43c111384846000001") }
46   //>
47   //

```

Listing 6.8: Mengakses mongoDB dari Node.js

6.4. Aplikasi Web Menggunakan Node.js dan mongoDB

Contoh aplikasi web berikut hanya digunakan untuk mengambil data dari mongoDB kemudian menampilkannya di web. Data diambil dari basis data mongoDB yang sudah dibuat sebelumnya (mydb). Untuk keperluan ini, kita akan menggunakan framework Express (<http://expressjs.com>). Install Express di level global dengan *npm install -g express*. Setelah terinstall, buat subdirektori baru (lokasi bebas) yang akan digunakan untuk menyimpan aplikasi web. Setelah itu, masuk ke direktori tersebut kemudian buat kerangka aplikasi di subdirektori tersebut menggunakan perintah “express” (lihat bab 1).

Berikut ini adalah beberapa perubahan yang dilakukan untuk rerangka aplikasi yang dihasilkan dari perintah *express* tersebut.

```

1
2  /**
3   * Module dependencies.
4   */
5
6  var express = require('express')
7    , routes = require('./routes')
8    , user = require('./routes/employee')
9    , http = require('http')
10   , path = require('path');

```

```

11
12 var app = express();
13
14 // all environments
15 app.set('port', process.env.PORT || 3000);
16 app.set('views', __dirname + '/views');
17 app.set('view engine', 'jade');
18 app.use(express.favicon());
19 app.use(express.logger('dev'));
20 app.use(express.bodyParser());
21 app.use(express.methodOverride());
22 app.use(app.router);
23 app.use(express.static(path.join(__dirname, 'public')));
24
25 // development only
26 if ('development' === app.get('env')) {
27   app.use(express.errorHandler());
28 }
29
30 app.get('/', routes.index);
31 app.get('/employees', employee.list);
32
33 http.createServer(app).listen(app.get('port'), function(){
34   console.log('Express server listening on port ' + app.get('port'));
35 });

```

Listing 6.9: Express+MongoDB web: app.js

```

1 {
2   "name": "show-employees",
3   "version": "0.0.1",
4   "private": true,
5   "scripts": {
6     "start": "node app.js"
7   },
8   "dependencies": {
9     "express": "3.2.2",
10    "jade": "*",
11    "mongojs": "latest"
12  }
13 }

```

Listing 6.10: Express+MongoDB web: package.json

```

1
2 /*
3  * GET home page.
4  */
5
6 exports.index = function(req, res){
7   res.render('index', { title: 'Contoh Express+mongoDB' });
8 };

```

Listing 6.11: Express+MongoDB web: routes/index.js

Selain itu, ada beberapa tambahan file (routes/employee.js dan views/employee.jade), penghapusan file (routes/user.js), dan perubahan yang cukup signifikan pada file *views/index.jade*.

```

1 i /*

```

```
2  * GET employees listing.
3  */
4
5  var databaseUrl = "localhost/mydb";
6  var collections = ["employees"];
7  var db = require("mongojs").connect(databaseUrl, collections);
8
9  exports.list = function(req, res){
10
11     // mencari dan menampilkan semua pegawai
12     db.employees.find(function(err, employees) {
13         res.render('employee', {listOfEmployee: employees, title: 'Daftar pegawai'});
14     });
15
16 };
```

Listing 6.12: Express+MongoDB web: routes/employee.js

```
1  extends layout
2
3  block content
4      hl= title
5      p #{title}
6
7      each employee in listOfEmployee
8          p #{employee.name}
```

Listing 6.13: Express+MongoDB web: views/employee.jade

```
1  extends layout
2
3  block content
4      hl= title
5      p
6          | Selamat datang di #{title}. Aplikasi ini hanya sekedar contoh
7          | aplikasi web dengan mongoDB sebagai backend. Untuk saat ini hanya
8          | tersedia fasilitas untuk melihat
9          a(href="/employees") daftar pegawai.
```

Listing 6.14: views/index.jade

Pola Arsitektur Aplikasi Web: MVC dan ExpressJS

7.1. Apa itu Pola Arsitektur?

Pola arsitektur (*architectural pattern*) adalah konsep dan standar arsitektur yang membentuk suatu aplikasi. Pola disini mengacu pada *best practices* atau praktik-praktik terbaik yang terutama terkait dengan arsitektur dari software aplikasi. Pola arsitektur terdiri atas elemen-elemen software, properti dari elemen-elemen tersebut, serta hubungan antar elemen-elemen tersebut.

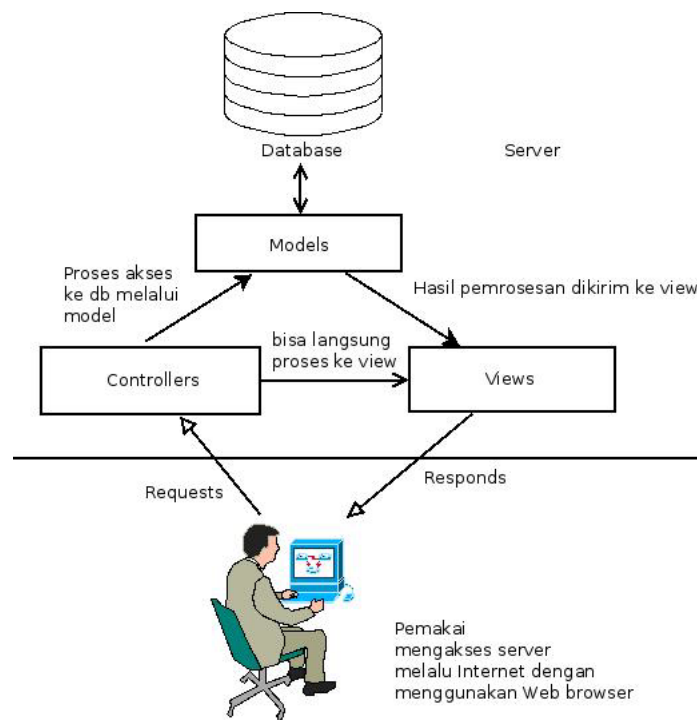
7.2. Pola Arsitektur MVC

MVC (Model-View-Controller) merupakan pola arsitektur aplikasi Web yang memisahkan aplikasi Web menjadi 3 komponen:

- ✓ Model: basis data
- ✓ View: tampilan antarmuka aplikasi Web, biasanya berisi semacam template dan isi-isi dinamis dari tampilan antarmuka tersebut.
- ✓ Controller: menerima *requests* atau permintaan dari browser kemudian mengarahkan ke *event-handler* untuk diproses. Proses tersebut bisa saja berupa langsung menghasilkan view (X)HTML atau format lainnya, atau bisa juga diproses terlebih dahulu di model dan kemudian hasilnya akan dikirimkan ke view untuk diisikan ke isi-isi dinamis serta membentuk file (X)HTML untuk ditampilkan di browser (sebenarnya tidak selalu perlu harus (X)HTML).

Jika digambarkan dalam suatu diagram, pola arsitektur MVC ditampilkan pada gambar [7.1](#)

Pola ini dikenal juga dengan istilah Model 2 dan dipopulerkan oleh JavaEE.



Gambar 7.1.: Pola arsitektur MVC

7.3. Implementasi Pola Arsitektur MVC Menggunakan ExpressJS

Sebenarnya ExpressJS bukan merupakan *framework* MVC, meskipun demikian karena framework ini sangat fleksibel, maka pemrogram bisa mengatur sendiri lokasi dari file / direktori serta berbagai konfigurasi lainnya. Contoh implementasi disini adalah aplikasi sederhana untuk menampilkan data yang tersimpan dalam basis data mongoDB ke dalam format JSON yang bisa diakses dari browser.

7.3.1. Struktur Aplikasi

Setelah membuat kerangka aplikasi menggunakan ExpressJS, ada beberapa perubahan yang harus dilakukan. Perubahan ini terutama dilakukan untuk mengikuti pola arsitektur MVC (terutama peletakan file dan direktori). Pola asli dari kerangka aplikasi ExpressJS adalah sebagai berikut:

```

1 $ tree .
2 .
3 |-- app.js
4 |-- package.json
5 |-- public
6 |   |-- images
7 |   |-- javascripts
8 |   |-- stylesheets
9 |   |-- style.css
10 |-- routes
11 |   |-- index.js
12 |   |-- user.js
13 |-- views
14 |   |-- index.jade
15 |   |-- layout.jade
16
17 6 directories , 7 files
18 $

```

Listing 7.1: Struktur direktori asli aplikasi ExpressJS

Struktur direktori tersebut akan diubah sesuai dengan pola MVC:

```

1 $ tree .
2 .
3 +-- app.js
4 |-- controllers
5 |   +-- index.js
6 |   +-- user.js
7 |-- models
8 |   +-- db.js
9 |-- package.json
10 |-- public
11 |   +-- images
12 |   |-- javascripts
13 |   +-- stylesheets
14 |   +-- style.css
15 +-- views
16 |   +-- index.jade
17 |   +-- layout.jade
18

```

```

19 7 directories , 8 files
20 $

```

Listing 7.2: Struktur direktori ExpressJS sesuai pola MVC

Beberapa perubahan terhadap struktur direktori:

- ✓ direktori routes diubah menjadi *controllers*
- ✓ membuat direktori *models* untuk mendefinisikan skema basis data

7.3.2. File-file yang Diperlukan

Beberapa file diubah isinya dan ada juga file yang baru.

```

1  /**
2   * Module dependencies.
3   */
4
5
6  var express = require('express')
7    , routes = require('./controllers')
8    , user = require('./controllers/user')
9    , http = require('http')
10    , path = require('path');
11
12  var mongoose = require('mongoose');
13
14  var app = express();
15
16  // all environments
17  app.set('port', process.env.PORT || 3000);
18  app.set('views', __dirname + '/views');
19  app.set('view engine', 'jade');
20  app.use(express.favicon());
21  app.use(express.logger('dev'));
22  app.use(express.bodyParser());
23  app.use(express.methodOverride());
24  app.use(app.router);
25  app.use(express.static(path.join(__dirname, 'public')));
26
27  // development only
28  if ('development' == app.get('env')) {
29    app.use(express.errorHandler());
30  }
31
32  app.get('/', controllers.index);
33  app.get('/users', user.list);
34
35  http.createServer(app).listen(app.get('port'), function(){
36    console.log('Express server listening on port ' + app.get('port'));
37  });

```

Listing 7.3: app.js

```

1  /**
2   * GET employees listing.
3   */
4
5  var Employee = require('../models/db.js');

```

```
6
7 exports.list = function(req, res){
8     Employee.find(function(err, employees) {
9         res.send(employees);
10    });
11 };
```

Listing 7.4: controllers/user.js

```
1 var mongoose = require('mongoose')
2   ,Schema = mongoose.Schema;
3
4 var employeeSchema = new Schema({
5     name: String,
6     address: String,
7     phone: String,
8     email: String
9 });
10
11 module.exports = mongoose.model('Employee', employeeSchema);
```

Listing 7.5: models/db.js

```
1 {
2   "name": "express-mvc",
3   "version": "0.0.1",
4   "private": true,
5   "scripts": {
6     "start": "node app.js"
7   },
8   "dependencies": {
9     "express": "latest",
10    "jade": "*",
11    "mongoose": "latest"
12  }
13 }
```

Listing 7.6: package.json

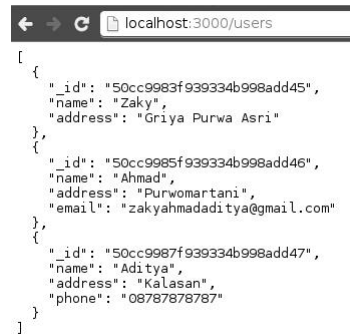
7.3.3. Hasil

Setelah server dieksekusi (menggunakan perintah *node app.js*), maka hasilnya akan bisa diakses di <http://localhost:3000/users>. Hasil di browser bisa dilihat di gambar 7.2

7.4. Pola Arsitektur Aplikasi Web Lain dan Implementasinya

MVC bukan satu-satu pola arsitektur aplikasi Web. Berikut ini adalah beberapa daftar pola arsitektur aplikasi Web serta implementasinya di Node.js dan/atau JavaScript di sisi klien:

- ✓ MVP (Model-View-Presenter): Google GWT.
- ✓ MVVM (Model-View-ViewModel): Batman.js (<http://batmanjs.org>) dan Knockout.js (<http://knockoutjs.com>)



A screenshot of a web browser window. The address bar shows 'localhost:3000/users'. The main content area displays a JSON array of three user objects. The first object has an ID, name 'Zaky', and address 'Griya Purwa Asri'. The second object has an ID, name 'Ahmad', address 'Purwomartani', and email 'zakyahmadaditya@gmail.com'. The third object has an ID, name 'Aditya', address 'Kalasan', and phone number '087878787'.

```
[
  {
    "_id": "50cc9983f939334b998add45",
    "name": "Zaky",
    "address": "Griya Purwa Asri"
  },
  {
    "_id": "50cc9985f939334b998add46",
    "name": "Ahmad",
    "address": "Purwomartani",
    "email": "zakyahmadaditya@gmail.com"
  },
  {
    "_id": "50cc9987f939334b998add47",
    "name": "Aditya",
    "address": "Kalasan",
    "phone": "087878787"
  }
]
```

Gambar 7.2.: Pola arsitektur MVC

- ✓ RVP (Resource-View-Presenter): Flatiron (<http://flatironjs.org>)
- ✓ MVA (Model-View-Adapter).
- ✓ Hierarchical MVC
- ✓ Presentation-Abstract-Control.

8.1. Apa itu Real-time Web?

Real-time Web menunjukkan suatu pola interaksi aplikasi Web yang memungkinkan kedua sisi saling mengirimkan data saat terjadi perubahan, jadi tidak seperti pola interaksi yang menghendaki pemakai untuk *me-refresh* browser jika menginginkan data / informasi / *update* terbaru dari sisi server. Contoh dari real-time Web adalah Facebook dan Twitter. Pemakai akan mendapatkan *update* secara langsung saat terjadi perubahan (komentar baru, pesan masuk, permintaan pertemanan, *retweet*, dan lain-lain), tanpa perlu *me-refresh* halaman.

8.2. Teknologi Pendukung Real-time Web

Real-time Web merupakan hal yang relatif kompleks. Terdapat beberapa teknologi yang bisa digunakan untuk mewujudkan real-time Web tersebut. Beberapa diantaranya merupakan standar (atau akan menjadi standar), sedangkan lainnya bukan merupakan standar.

8.2.1. *Ajax Technology*

Teknologi Ajax (kadang juga ditulis AJAX, singkatan dari *Asynchronous JavaScript and XML*) adalah sekumpulan teknologi yang pertama kali dicetuskan oleh Jesse James Garrett. Ajax memungkinkan browser untuk mengirim data dan mengambil data dari server secara *asynchronous* (di latar belakang) tanpa mengganggu keseluruhan tampilan halaman Web. Kumpulan teknologi yang digunakan adalah:

- ✓ (X)HTML dan CSS untuk presentasi halaman Web

- ✓ DOM (*Document Object Model*) untuk menampilkan data secara dinamis
- ✓ XML dan XSLT untuk pertukaran data (seringkali tidak menggunakan XML tetapi JSON).
- ✓ Obyek XMLHttpRequest untuk komunikasi asynchronous
- ✓ JavaScript

8.2.2. Comet dan *Push Technology*

Comet merupakan istilah payung yang merangkum berbagai teknologi *push*, yaitu teknologi yang memungkinkan server untuk mengirimkan data ke browser tanpa diminta oleh browser.

SSE (*Server-Sent Events*)

SSE merupakan bagian dari spesifikasi standar HTML5 (bisa diakses di <http://dev.w3.org/html5/eventsource/>). Spesifikasi ini memungkinkan server untuk mem-*push* data ke halaman Web menggunakan protokol HTTP. Meski masih dalam pengembangan, tetapi beberapa browser sudah mendukung (misalnya Google Chrome / Chromium) serta Safari. Beberapa peranti pengembangan di sisi server juga sudah mendukung spesifikasi ini. Pada Node.js, pemrogram bisa menggunakan paket sse, nsse, atau EventSource.

Bayeux Protocol

Protokol ini dikembangkan oleh *the Dojo Foundation* yang mengembangkan software Dojo Toolkit. Protokol ini digunakan sebagai transport untuk pesan-pesan asynchronous melalui HTTP dengan latensi yang rendah antara klien dengan server. Pesan-pesan tersebut di-rute-kan melalui channel-channel yang diberi nama dan bisa dikirimkan ke:

- ✓ server ke klien
- ✓ klien ke server
- ✓ klien ke klien (melalui server)

Spesifikasi lengkap dari protokol ini bisa dilihat di <http://svn.cometd.com/trunk/bayeux/bayeux.html>.

BOSH Protocol

BOSH (Bidirectional-streams Over Synchronous HTTP) adalah protokol transport yang mengemulasi stream dua arah antara dua entitas (misalnya antara klien dengan server) dengan menggunakan banyak HTTP req/resp yang synchronous tanpa memerlukan polling yang sering atau respon yang terpotong-potong. Spesifikasi ini dikembangkan oleh komunitas serta yayasan XMPP dan bisa dilihat secara lengkap di <http://xmpp.org/extensions/xep-0124.html>

8.2.3. WebSocket

WebSocket merupakan teknologi Web yang menyediakan saluran komunikasi full duplex pada satu koneksi TCP. Protokol WebSocket distandarkan oleh IETF di RFC 6455 sedangkan API (*Application Programming Interface*) dikembangkan dan distandarkan oleh W3C sebagai bagian dari HTML5. Komunikasi antara klien dengan server dilaksanakan menggunakan TCP dengan nomor port 80.

WebSocket diimplementasikan di sisi server dan klien dan memungkinkan adanya interaksi yang lebih real-time daripada teknologi push karena protokol dan API ini diimplementasikan dan bisa digunakan di sisi klien maupun server. Browser yang sudah mendukung protokol dan API WebSocket ini adalah Chrome, Firefox, Safari, Opera, dan Internet Explorer.

Perkembangan dari WebSocket bisa dilihat dan diikuti di <http://www.websocket.org/>

8.3. Socket.io

8.3.1. Apa itu Socket.io?

Socket.io adalah pustaka JavaScript yang merupakan implementasi dari protokol WebSocket serta berbagai improvisasi lain yang diperlukan untuk real-time web (*heartbeats*, *timeouts*, dan *disconnection*). Protokol transport yang didukung adalah sebagai berikut:

- ✓ WebSocket
- ✓ Adobe Flash Socket
- ✓ AJAX long polling
- ✓ AJAX multipart streaming
- ✓ Forever Iframe
- ✓ JSONP Polling

Pustaka ini terdiri atas pustaka untuk sisi klien (browser) dan server (menggunakan Node.js). Browser yang didukung adalah:

- ✓ Internet Explorer 5.5+ (desktop)
- ✓ Safari 3+ (desktop)
- ✓ Google Chrome 4+ (desktop)
- ✓ Firefox 3+ (desktop)
- ✓ Opera 10.61+ (desktop)
- ✓ iPhone Safari (mobile)

- ✓ iPad Safari (mobile)
- ✓ Android WebKit (mobile)
- ✓ WebOs WebKit (mobile)

8.3.2. Menggunakan Socket.io untuk Real-time Web

Socket.io melibatkan sisi klien dan sisi server. Pada sisi server, paket yang diperlukan adalah `socket.io`, sementara untuk sisi klien (browser), diperlukan `socket.io-client`. Paket `socket.io-client` tidak diperlukan langsung pada sisi `node_modules`, tetapi ada beberapa file yang harus ditempatkan pada akses publik dengan maksud supaya bisa digunakan oleh browser.

Tentang Aplikasi

Aplikasi ini hanya merupakan contoh kecil dari real-time Web. Aplikasi terdiri atas sisi server dan klien/browser. Pada sisi server, aplikasi ini akan mengirimkan data ke browser (push). Sementara itu, browser akan menerima hasil push tersebut dan menampilkannya kemudian mengirimkan data ke server tanpa perlu melakukan proses *refresh*. Server hanya akan menampilkan data yang dikirimkan browser.

Membuat Kerangka Aplikasi dengan ExpressJS

Untuk membuat aplikasi ini, kita akan menggunakan ExpressJS dan Socket.io. Pada awalnya, kita akan membuat kerangka aplikasi menggunakan `express` (jika ExpressJS belum terinstall, install dengan menggunakan `npm install -g express`). Jika sudah terinstall, buat direktori baru, kemudian buatlah kerangka aplikasi menggunakan `express` pada direktori tersebut: `express<Enter>`.

Pada pembahasan berikutnya, kita akan mengadakan berbagai perubahan yang diperlukan.

Instalasi Paket yang Diperlukan

File `package.json` berisi beberapa informasi tentang aplikasi ini serta beberapa paket yang diperlukan. Isi dari file ini adalah sebagai berikut:

```
1 {  
2   "name": "socket.io-expressjs",  
3   "version": "0.0.1",  
4   "author": "Bambang Purnomosidi D. P.",  
5   "private": true,  
6   "scripts": {  
7     "start": "node app"  
8   },  
9   "dependencies": {  
10    "express": "latest",  
11    "jade": "*",  
12    "socket.io": "latest",  
13    "socket.io-client": "latest"
```

```

14 }
15 }

```

Listing 8.1: package.json

Setelah itu, install paket-paket tersebut dengan menggunakan perintah *npm install* di direktori tersebut.

Konfigurasi JavaScript untuk Browser

Browser juga memerlukan pustaka untuk Socket.io yang diperoleh dari paket *socket.io-client*. Pada paket tersebut, terdapat direktori *dist*:

```

1 $ ls -la node_modules/socket.io-client/dist/
2 total 496
3 drwxr-xr-x 2 bdp bdp 4096 May 6 00:00 .
4 drwxr-xr-x 7 bdp bdp 4096 May 6 00:01 ..
5 -rw-r--r-- 1 bdp bdp 101222 Nov 2 2012 socket.io.js
6 -rw-r--r-- 1 bdp bdp 44789 Nov 2 2012 socket.io.min.js
7 -rw-r--r-- 1 bdp bdp 175953 Mar 28 2012 WebSocketMainInsecure.swf
8 -rw-r--r-- 1 bdp bdp 175830 Mar 28 2012 WebSocketMain.swf
9 $

```

Listing 8.2: Isi direktori dist di socket.io-client

Copy-kan file-file tersebut ke direktori *public/javascripts*.

Hapus File yang Tidak Diperlukan

Ada beberapa file yang tidak diperlukan dan harus dihapus:

✓ routes/user.js

Ubah File-file Tertentu

Beberapa file akan diedit. Beberapa diantaranya akan diuraikan di bagian ini.

```

1
2 /**
3  * Module dependencies.
4  */
5
6 var express = require('express')
7   , app = express()
8   , routes = require('./routes')
9   , http = require('http')
10  , path = require('path')
11  , server = http.createServer(app)
12  , io = require('socket.io').listen(server);
13
14 server.listen(80);
15
16 // all environments
17 app.set('views', __dirname + '/views');
18 app.set('view engine', 'jade');
19 app.use(express.favicon());
20 app.use(express.logger('dev'));

```

```

21 app.use(express.bodyParser());
22 app.use(express.methodOverride());
23 app.use(app.router);
24 app.use(express.static(path.join(__dirname, 'public')));
25
26 app.get('/', routes.index);
27
28 io.sockets.on('connection', function (socket) {
29     socket.emit('kirim ke browser', {
30         kalimatDariServer: 'Kalimat ini dikirim dari server' });
31     socket.on('dari browser', function (data) {
32         console.log(data.kalimatDariBrowser);
33     });
34 });

```

Listing 8.3: app.js

```

1 extends layout
2
3 block content
4     h1= title
5     p #{title}
6     script(src="/javascripts/socket.io.js")
7     script
8         var socket = io.connect('http://localhost');
9         socket.on('kirim ke browser', function (data) {
10             document.getElementById("container").innerHTML=
11                 "<p>" + data.kalimatDariServer + "</p>";
12             socket.emit('dari browser', {
13                 kalimatDariBrowser: 'Kalimat ini dikirim dari browser' });
14         });
15     #container
16     p #{title}

```

Listing 8.4: views/index.jade

```

1 /*
2  * GET home page.
3  */
4
5 exports.index = function(req, res){
6     res.render('index', { title: 'Contoh Socket.io + Express' });
7 };

```

Listing 8.5: routes/index.js

Menjalankan Server Socket.io

Server socket.io menggunakan port 80 sehingga harus dijalankan oleh *root*.

```

1 info - socket.io started
2 GET / 200 141ms - 628b
3 GET /stylesheets/style.css 304 3ms
4 GET /javascripts/socket.io.js 200 9ms - 98.85kb
5 debug - client authorized
6 info - handshake authorized LOoIIHJJjO1lg34JtLnP
7 debug - setting request GET /socket.io/1/websocket/LOoIIHJJjO1lg34JtLnP
8 debug - set heartbeat interval for client LOoIIHJJjO1lg34JtLnP
9 debug - client authorized for

```

```

10  debug - websocket writing 1::
11  debug - websocket writing 5:::{"name":" kirim ke browser","args":
12  [{"kalimatDariServer":"Kalimat ini dikirim dari server"}]}
13  Kalimat ini dikirim dari browser
14  GET / 200 4ms - 628b
15  info - transport end (socket end)
16  debug - set close timeout for client LOoIHJJjO1lg34JtLnP
17  debug - cleared close timeout for client LOoIHJJjO1lg34JtLnP
18  debug - cleared heartbeat interval for client LOoIHJJjO1lg34JtLnP
19  debug - discarding transport
20  GET /stylesheets/style.css 304 1ms
21  GET /javascripts/socket.io.js 304 1ms
22  debug - client authorized
23  info - handshake authorized ijvgksAoa-BWm7uytLnQ
24  debug - setting request GET /socket.io/1/websocket/ijvgksAoa-BWm7uytLnQ
25  debug - set heartbeat interval for client ijvgksAoa-BWm7uytLnQ
26  debug - client authorized for
27  debug - websocket writing 1::
28  debug - websocket writing 5:::{"name":" kirim ke browser","args":
29  [{"kalimatDariServer":"Kalimat ini dikirim dari server"}]}
30  Kalimat ini dikirim dari browser
31  debug - emitting heartbeat for client ijvgksAoa-BWm7uytLnQ
32  debug - websocket writing 2::
33  debug - set heartbeat timeout for client ijvgksAoa-BWm7uytLnQ
34  debug - got heartbeat packet
35  debug - cleared heartbeat timeout for client ijvgksAoa-BWm7uytLnQ
36  debug - set heartbeat interval for client ijvgksAoa-BWm7uytLnQ

```

Listing 8.6: Menjalankan server Socket.io

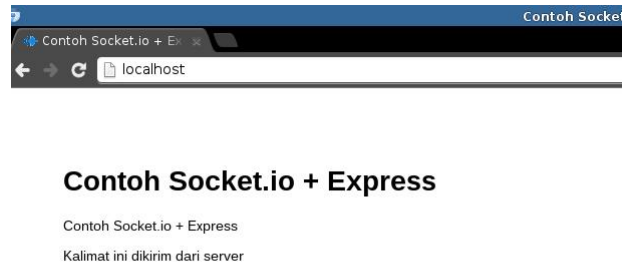
Keluaran pada sisi server tersebut merupakan keluaran yang sudah termasuk akses dari browser. Setelah server dijalankan, buka browser kemudian akses URL <http://localhost>. Setelah diakses melalui browser, server akan mengirimkan kode sumber HTML sebagai berikut:

```

1  <!DOCTYPE html>
2  <html>
3      <head>
4          <title>Contoh Socket.io + Express</title>
5          <link rel="stylesheet" href="/stylesheets/style.css">
6      </head>
7      <body>
8          <h1>Contoh Socket.io + Express</h1>
9          <p>Contoh Socket.io + Express</p>
10         <script src="/javascripts/socket.io.js"></script>
11         <script>
12             var socket = io.connect('http://localhost');
13             socket.on(' kirim ke browser', function (data) {
14                 document.getElementById("container").innerHTML=
15                 "<p>" + data.kalimatDariServer + "</p>";
16                 socket.emit('dari browser', {
17                     kalimatDariBrowser: 'Kalimat ini dikirim dari browser' });
18             });
19         </script>
20         <div id="container">
21             <p>Contoh Socket.io + Express</p>
22         </div>
23     </body>
24 </html>

```

Listing 8.7: Kode sumber di browser



Gambar 8.1.: Hasil di browser dari ExpressJS + Socket.io

Tampilan di browser bisa dilihat pada gambar [8.1](#)

Contoh pada materi ini merupakan contoh sederhana, tetapi diharapkan bisa dengan mudah dipahami untuk membuat aplikasi Web real-time.

- [1] Anonim, *Mozilla Developer Network - JavaScript*, <https://developer.mozilla.org/en-US/docs/JavaScript>.
- [2] Cody Lindley, *JavaScript Enlightenment*, <http://javascriptenlightenment.com>, 2012.
- [3] David Flanagan, *JavaScript: The Definitive Guide*, 4th Edition, O'Reilly, 2001.
- [4] Don Nguyen, *Jump Start Node.js*, SitePoint, 2012.
- [5] Douglas Crockford, *JavaScript: The Good Parts*, O'Reilly, 2008.
- [6] Marijn Haverbeke, *Eloquent JavaScript: A Modern Introduction to Programming*, No Starch Press, 2011.
- [7] Shelley Powers, *Learning Node*, O'Reilly, 2012.
- [8] Tom Hughes-Croucher, Mike Wilson, *Node: Up and Running*, O'Reilly, 2012.

Lampiran

Gaya Penulisan Kode / *Coding Style*

A.1. Tentang Gaya Penulisan Kode

Pada saat seseorang berada pada proses pembuatan kode sumber (*coding*), apapun gaya penulisan programnya, tidak akan bermasalah jika program itu hanya dia buat untuk si pemrogram itu sendiri. Meski demikian, hal tersebut jarang terjadi karena biasanya selalu ada kerja kelompok atau setidaknya program tersebut akan digunakan oleh pihak lain yang suatu saat perlu memahami apa yang tertulis di kode sumber tersebut. Untuk keperluan itu, biasanya diperlukan suatu gaya penulisan kode. Saat ini banyak sekali gaya penulisan kode untuk JavaScript / Node.js, biasanya tergantung pada kesepakatan anggota-anggota dalam kelompok pengembang dan berdasarkan pada pengalaman mereka tersebut. Pada lampiran ini, gaya penulisan kode dari NPM akan dituliskan secara utuh (dalam bahasa aslinya dengan harapan bisa bermanfaat untuk penyeragaman dan kemudahan membaca atau menelusuri *bugs/errors*). Gaya penulisan kode ini diambil dari <https://npmjs.org/doc/coding-style.html>.

A.2. npm's Coding Style

npm's "funny" coding style

A.2.1. DESCRIPTION

npm's coding style is a bit unconventional. It is not different for difference's sake, but rather a carefully crafted style that is designed to reduce visual clutter and make bugs more apparent.

If you want to contribute to npm (which is very encouraged), you should make your code conform to npm's style.

A.2.2. Line Length

Keep lines shorter than 80 characters. It's better for lines to be too short than to be too long. Break up long lists, objects, and other statements onto multiple lines.

A.2.3. Indentation

Two-spaces. Tabs are better, but they look like hell in web browsers (and on github), and node uses 2 spaces, so that's that.

Configure your editor appropriately.

A.2.4. Curly braces

Curly braces belong on the same line as the thing that necessitates them. Bad:

```
1 function ()  
2 {
```

Listing A.1: Bad curly braces placement - 1

Good:

```
1 function () {
```

Listing A.2: Good curly braces placement - 1

If a block needs to wrap to the next line, use a curly brace. Don't use it if it doesn't. Bad:

```
1 if (foo) { bar() }  
2 while (foo)  
3   bar()
```

Listing A.3: Bad curly braces placement - 2

Good:

```
1 if (foo) bar()  
2 while (foo) {  
3   bar()  
4 }
```

Listing A.4: Good curly braces placement - 2

A.2.5. Semicolons

Don't use them except in four situations:

- ✓ **for (;) loops.** They're actually required.
- ✓ null loops like: **while (something) ;** (But you'd better have a good reason for doing that.)
- ✓ **case "foo": doSomething(); break**

- ✓ In front of a leading (or [at the start of the line. This prevents the expression from being interpreted as a function call or property access, respectively.

Some examples of good semicolon usage:

```
1 ;(x || y).doSomething()  
2 ;[a, b, c].forEach(doSomething)  
3 for (var i = 0; i < 10; i ++) {  
4   switch (state) {  
5     case "begin": start(); continue  
6     case "end": finish(); break  
7     default: throw new Error("unknown state")  
8   }  
9   end()  
10 }
```

Listing A.5: Good semicolon usage

Note that starting lines with - and + also should be prefixed with a semicolon, but this is much less common.

A.2.6. Comma First

If there is a list of things separated by commas, and it wraps across multiple lines, put the comma at the start of the next line, directly below the token that starts the list. Put the final token in the list on a line by itself. For example:

```
1 var magicWords = [  
2   "abracadabra"  
3   , "gesundheit"  
4   , "ventrilo"  
5   ]  
6   , spells = { "fireball" : function () { setOnFire() }  
7   , "water" : function () { putOut() }  
8   }  
9   , a = 1  
10  , b = "abc"  
11  , etc  
    , somethingElse
```

Listing A.6: Comma first

A.2.7. Whitespace

Put a single space in front of (for anything other than a function call. Also use a single space wherever it makes things more readable.

Don't leave trailing whitespace at the end of lines. Don't indent empty lines. Don't use more spaces than are helpful.

A.2.8. Functions

Use named functions. They make stack traces a lot easier to read.

A.2.9. Callbacks, Sync/async Style

Use the asynchronous/non-blocking versions of things as much as possible. It might make more sense for npm to use the synchronous fs APIs, but this way, the fs and http and child process stuff all uses the same callback-passing methodology.

The callback should always be the last argument in the list. Its first argument is the Error or null.

Be very careful never to ever ever throw anything. It's worse than useless. Just send the error message back as the first argument to the callback.

A.2.10. Errors

Always create a new Error object with your message. Don't just return a string message to the callback. Stack traces are handy.

A.2.11. Logging

Logging is done using the npmlog utility.

Please clean up logs when they are no longer helpful. In particular, logging the same object over and over again is not helpful. Logs should report what's happening so that it's easier to track down where a fault occurs.

Use appropriate log levels. See `config(1)` and search for "loglevel".

A.2.12. Case, naming, etc.

- ✓ Use **lowerCamelCase** for multiword identifiers when they refer to objects, functions, methods, members, or anything not specified in this section.
- ✓ Use **UpperCamelCase** for class names (things that you'd pass to "new").
- ✓ Use **all-lower-hyphen-css-case** for multiword filenames and config keys.
- ✓ Use named functions. They make stack traces easier to follow.
- ✓ Use **CAPS_SNAKE_CASE** for constants, things that should never change and are rarely used.
- ✓ Use a single uppercase letter for function names where the function would normally be anonymous, but needs to call itself recursively. It makes it clear that it's a "throwaway" function.

A.2.13. null, undefined, false, 0

- ✓ Boolean variables and functions should always be either **true** or **false**. Don't set it to 0 unless it's supposed to be a number.
- ✓ When something is intentionally missing or removed, set it to **null**.

- ✓ Don't set things to **undefined**. Reserve that value to mean "not yet set to anything."
 - ✓ Boolean objects are verboten.
-

Selain gaya penulisan kode dari NPM ini, ada beberapa lagi lainnya yang bisa dilihat, antara lain:

- ✓ Spludo <http://spludo.com/source/coding-standards/>
- ✓ Google JavaScript Style Guide (<http://google-styleguide.googlecode.com/svn/trunk/javascriptguide.xml>)
- ✓ Felix's Node.js Style Guide (<http://nodeguide.com/style.html>)

Commit History dari Penulis Utama dan Kontributor

Buku ini merupakan hasil karya bersama dari beberapa penulis. Peran masing-masing penulis bisa dilihat pada bagian ringkasan dari sejarah *commit*. Penulis utama adalah saya (Bambang Purnomosidi D. P), sementara pada bab 5 ada kontribusi dari Aji Kisworo Mukti. Hasil log dari git menunjukkan peran masing-masing penulis (*git shortlog*):

```

1 Aji Kisworo Mukti (3):
2   Bab 5 – Struktur Instalasi Paket Node.js
3   Bab 5 – Instalasi Paket
4   Bab 5 – Menghapus Paket
5
6 Bambang Purnomosidi D. P (77):
7   Initial commit
8   First commit – initializing empty repo
9   Merge branch 'master' of https://github.com/bpdp/buku-cloud-nodejs
10  Menambahkan link ke teks bahasa Indonesia untuk lisensi CC-BY-SA
11  Menambahkan link ke teks bahasa Indonesia untuk lisensi CC-BY-SA
12  Menambahkan link ke teks bahasa Indonesia untuk lisensi CC-BY-SA
13  Menambahkan tips untuk indeks
14  Melengkapi bab 1, terutama tentang teori Cloud Computing
15  kesalahan kecil, tidak menutup textit dengan { tapi |
16  Menambahkan indeks dari Bab 1
17  Bab 1 selesai
18  Bab 2 – bagian REPL selesai
19  Edit bagian instalasi Flatiron – hasil direktori
20  Menambahkan tentang penulis buku
21  Bab 1 – sedikit keterangan ttg Node.js, Bab 2 – awal dasar2 JavaScript
22  Penambahan isi di bab 2 dan 7
23  Update bab 5 -> mengubah NPM mjd npm dan menambahkan 'Apakah npm itu?'
24  Makefile => buat clean-all dan clean-without-pdf, bab 2 selesai Readline
25  trivial changes
26  Penambahan di bab 2, menetapkan shadowbox untuk 'catatan'
27  Bab 2: nilai, tipe data, dan variabel. Menambahkan utk catatan ke tips
28  Penambahan tentang Literal dan reorganisasi sub bab (fungsi)
29  Bab 2: Pembahasan 'Fungsi' selesai.
30  Bab 2 – Literal, selesai
31  Bab 2 – Pernyataan kondisi if .. else if .. else: selesai
32  Bab 2 – JSON, switch, dan looping for — selesai

```

```

33     Memperbaiki sedikit typo, kurang satu { di footnote wikipedia utk JSON
34     Bab 2 selesai
35     Bab 3 – pengertian PBO dan definisi obyek -> selesai
36     Menambahkan Aji Kisworo Mukti ke kontributor di README.md
37     Minor revision di bab 5, menambahkan gambar npmls (soalnya kode ASCII
38     keluarannya kacau di LaTeX dan saya blm tau workaround-nya
39     Mengubah cover -> lebih umum, ganti dg logo NodeJS, menambahkan Aji ke
40     kontributor, appendix B -> commit hist dari kontributor
41     Menambahkan materi PBO => melengkapi definisi obyek serta inheritance.
42     Contoh inheritance.js ditambahkan
43     Bab 3 – Pemrograman fungsional di JS => pengertian + beberapa point yg
44     akan dibahas
45     Menambahkan info tentang koma-script di README.md dan Makefile versi
46     terakhir
47     Menambahkan nested functions di bab 2
48     Menambahkan source code nested.js (bab 2)
49     Bab 3 – beberapa penambahan di pemrograman fungsional
50     Bab 3 – Lambda Expression + contoh
51     Higher-order function – Bab 3
52     Menyelesaikan Closure dan Currying di Bab 3. Bab 3 sudah selesai.
53     Mengganti struktur – bab 4 -> 5 dan sebaliknya. Bab 4 selesai, Bab 3
54     minor rev
55     Bab 5 -> (A) Synchronous programming
56     Bab 5 -> reorganisasi, minor revision
57     Bab 5: Event-Driven Programming menggunakan events.EventEmitter.
58     => Bab 5 selesai
59     Bab 6: Sedikit penjelasan tentang db NoSQL
60     Bab 6 – menambahkan penjelasan ttg mongoDB: fitur, server, client web
61     Bab 6: node-gyp dan instalasi driver mongodb
62     Bab 6: menambahkan instalasi npm untuk mongodb
63     Bab 6: install mongojs, akses mongojs dari Node.js. Kurang aplikasi web
64     Bab 4: menambahkan info ttg install ke homedir (jika berada dlm home)
65     dan node_modules (jika di luar home)
66     Bab 6: memulai aplikasi web dengan nodejs+expressjs+mongodb
67     Bab 6: src code utk aplikasi web nodejs+express+mongoDB selesai
68     Bab 6 selesai
69     Reorganisasi bab 7 dan 8, menghapus db mongoDB, menambahkan README.md
70     utk latihan2 di bab 6
71     Edit README.md bab 6
72     Menambah isi bab 7 dan 8
73     Bab 8: source code utk socket.io
74     Bab 8 selesai, menambahkan contoh aplikasi Socket.io
75     Selesai. sedikit pembenahan. hari ini bab 7 dan 8 selesai
76     Revisi minor bab 6 dan 8
77     Menambahkan daftar pustaka yang digunakan
78     Revisi minor di bbrp bab, terutama terkait margin kanan yg terlalu
79     bablas
80     Menambahkan indeks
81     Edit Appendix B (history commit) dan README.md (menambah status –
82     buku sudah selesai)
83     Menambahkan link ke file PDF di README.md
84     Memperbaharui sesuai dengan versi software terbaru (7 Jan 2013)+kata
85     pengantar
86     update README.md untuk merefleksikan kondisi terbaru
87     Cover: menambah lisensi dan lambang, Bab 1 diubah menyesuaikan dgn
88     Node 0.10.0 dan vmc 0.5.0
89     Menyesuaikan dengan semua versi software tgl 22 Maret 2013
90     Menambahkan status di README.md (as of March 22, 2013), memasukkan
91     commit history
92     Edit README.md – minor rev
93     listing dan tampilan2 teks layar skrg diambil dengan \lstinputlisting –
94     bab 1 selesai

```

```
95     Bab 2 selesai di-migrasi lstinputlisting , rename dir utk konsistensi
96     di src/
97     Mengedit src/README.md agar sesuai kondisi saat ini
98     Memperbarui semua versi software as of May 6, 2013. Clean up junkies ,
99     typos. Semua source code dan tampilan.txt dipisahkan dari file2
100     LaTeX utk modularitas
101     Lihat Changelogs.txt – tanggal 6 Mei 2013
102
103 Bambang Purnomosidi D. P. (1):
104     Merge pull request #1 from adzymaniac/master
```

Listing B.1: *Commit history*

- AJAX, [68](#)
- Aliran kendali, [31](#)
- Array, [28](#)
- Asynchronous, [50](#)

- Bayeux Protocol, [69](#)
- Boolean, [29](#)
- BOSH, [69](#)

- Callback, [50](#)
- Closure, [39](#)
- Cloud Computing, [1](#)
 - Definisi, [1](#)
 - Karakteristik, [1](#)
 - Model layanan, [3](#)
 - Private, [3](#)
 - Public, [3](#)
- CloudFoundry, [4](#)
 - vmc, [4](#)
- Comet, [69](#)
- CommonJS, [23](#)
- Currying, [39](#)

- Dynamically typed, [24](#)

- ECMAScript, [23](#)
- Ekspresi Lambda, [37](#)
- Event-Driven, [49](#)
- events.EventEmitter, [49](#)

- Floating-point, [29](#)

- Fungsi, [26](#)
- Fungsi Anonim, [27](#)
- Fungsi rekursif, [27](#)

- Higher-order Function, [38](#)

- Identifier, [25](#)
- if, [31](#)
- Integer, [29](#)

- JSON, [30](#)
 - Validator, [48](#)

- Konstanta, [25](#)

- Literal, [28](#)
- Looping, [33](#)

- mongodb, [52](#)
 - Driver, [57](#)
- Multitenancy, [3](#)
- MVC, [62](#)
 - dan ExpressJS, [64](#)

- Nested functions, [27](#)
- Node-gyp, [57](#)
- Node.js, [4](#)
 - Hosting, [4](#)
- Non-blocking IO, [50](#)
- NOSQL, [52](#)
- npm, [43](#)
 - Cari paket, [45](#)

- Hapus paket, [45](#)
- Info paket, [46](#)
- install paket, [44](#)
- Struktur paket, [44](#)
- Update paket, [47](#)

Obyek, [29](#), [40](#)

PBO, [40](#)

Penanganan error, [35](#)

Pewarisan, [41](#)

Pola arsitektur, [62](#)

Readline, [23](#)

Real-time Web, [68](#)

REPL, [21](#)

Server-Sent Events, [69](#)

Socket.io, [70](#)

String, [29](#)

switch, [32](#)

Tipe data, [24](#)

Variabel, [25](#)

WebSocket, [70](#)