Pengembangan Aplikasi Cloud Computing Menggunakan Node.js

Bambang Purnomosidi D. P. <a.out@bpdp.xyz>

Version 1.0, 2015-10

Table of Contents

Buku ini saya dedikasikan untuk keluarga saya dan guru-guru saya (anda mungkin salah satu diantaranya!). Buku bebas dengan lisensi CC-BY-SA ini merupakan buku yang dirancang untuk keperluan memberikan pengetahuan mendasar tentang 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. Versi sebelum 0.9.0 menggunakan *libev* dari Mark Lechmann.

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 DistroWatch. Sistem operasi Linux ini bukan keharusan, anda bisa menggunakan Windows tetapi silahkan membuat penyesuaian-penyesuaian sendiri yang diperlukan. Kirimi saya *pull request* jika anda menuliskan pengalaman anda di Windows!
- Git (untuk *version control system*) - bisa diperoleh di http://git-scm.com.
- mongoDB (basis data NOSQL) - bisa diperoleh di http://www.mongodb.org
- 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 Heroku), akan dibahas pada 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).

Have fun!

Pengenalan Cloud Computing dan Infrastruktur Pengembangan Aplikasi Berbasis Node.js

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: http://en.wikipedia.org/wiki/Cloud_computing].
- **NIST (The National Institute of Standards and Technology)**: 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 layanan. [2: http://csrc.nist.gov/publications/PubsSPs.html#800-145]

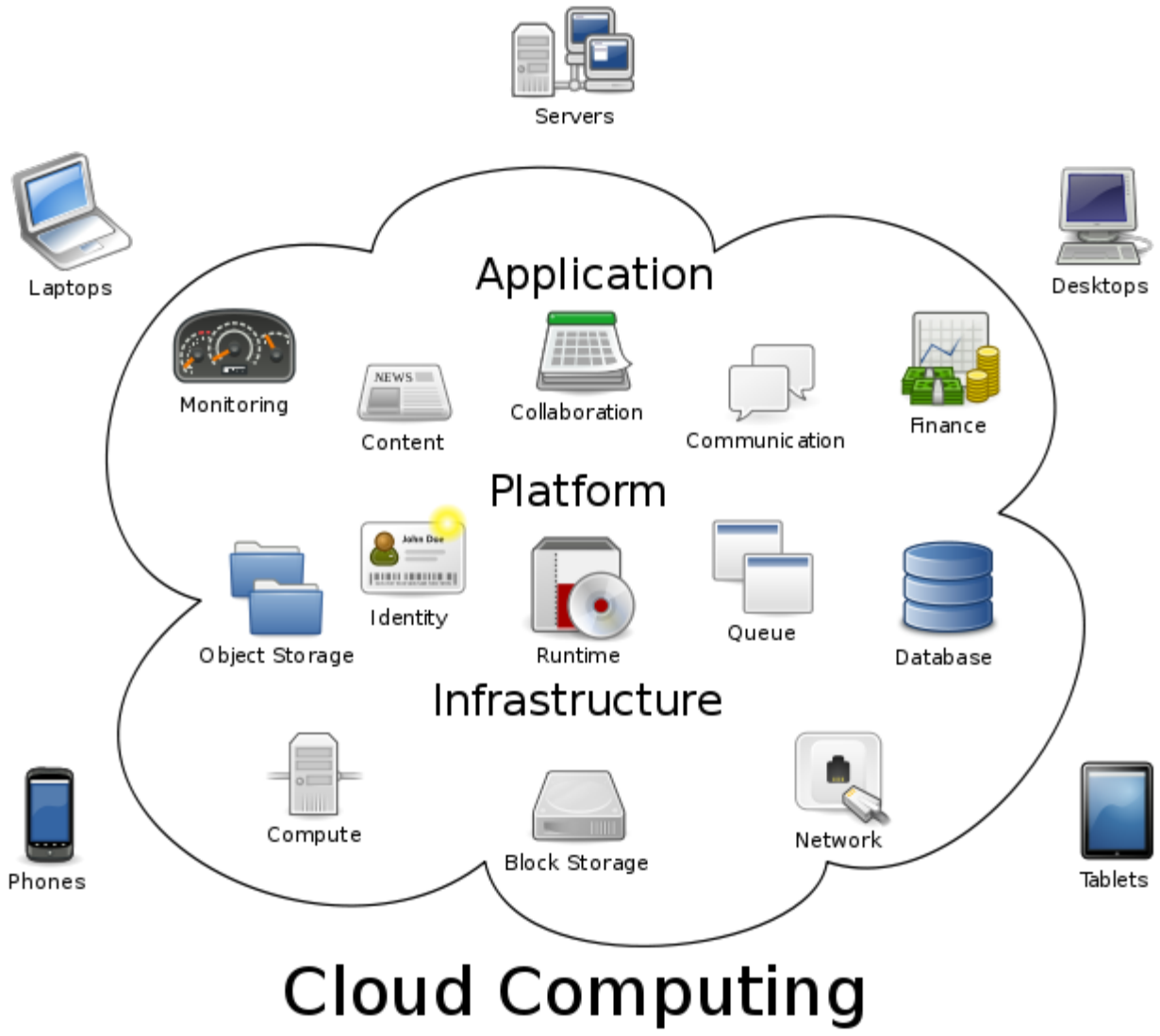

Figure 1. Model dari Cloud Computing

Jika diwujudkan secara visual, Cloud Computing bisa dilihat pada Model dari Cloud Computing. [3: Gambar dibuat oleh Sam Johnston, diambil dari http://en.wikipedia.org/w/index.php?title=File:Cloud_computing.svg&page=1]

Karakteristik Cloud Computing

Menurut NIST, ada beberapa karakteristik dari 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 tersebut. [4: http://en.wikipedia.org/wiki/Multitenancy]

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.

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 (pemrosesam, 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*).

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:

- *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).

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. Saat ini, Node.js dikembangkan oleh komunitas sebagai software bebas dibawah *Node.js foundation* yang merupakan suatu yayasan dibawah *Linux Foundation*.

REPL dan Dasar-dasar JavaScript di Node.js

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.

Mengaktifkan REPL

Untuk mengaktifkan REPL dari Node.js, *executable command line program*-nya adalah **node** (atau **nodejs** pada beberapa sistem operasi, misalnya Ubuntu Linux). Jika **node** dipanggil dengan argumen nama file JavaScript, maka file JavaScript tersebut akan dieksekusi, sementara jika tanpa argumen, akan masuk ke REPL:

```
$ node
>
```

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

```
> .help
break   Sometimes you get stuck, this gets you out
clear   Alias for .break
exit    Exit the repl
help    Show repl options
load    Load JS from a file into the REPL session
save    Save all evaluated commands in this REPL session to a file
>
```

Untuk keluar dari mode REPL, tekan **Ctrl-D** sekali atau **Ctrl-C** dua kali.

```
$ node
>
(^C again to quit)
>
```

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**:

```
$ node
> .load /home/bpdp/kerjaan/src/javascript/nodejs/hello.js
> var http = require('http');
undefined
> http.createServer(function (req, res) {
...   res.writeHead(200, {'Content-Type': 'text/plain'});
...   res.end('Hello World\n');
... }).listen(1337, '127.0.0.1');
{ domain: null,
  _events:
   { request: [Function],
     connection: [Function: connectionListener],
     clientError: [Function] },
  _maxListeners: 10,
  _connections: 0,
  connections: [Getter/Setter],
  _handle: null,
  _usingSlaves: false,
  _slaves: [],
  allowHalfOpen: true,
  httpAllowHalfOpen: false,
  timeout: 120000 }
> console.log('Server running at http://127.0.0.1:1337/');
Server running at http://127.0.0.1:1337/
undefined
>
```

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:

```
$ node
> console.log("Selamat datang di Node.js")
Selamat datang di Node.js
undefined
> .save /home/bpdp/kerjaan/src/javascript/nodejs/welcome.js
Session saved to:/home/bpdp/kerjaan/src/javascript/nodejs/welcome.js
> exit
$ cat /home/bpdp/kerjaan/src/javascript/nodejs/welcome.js
console.log("Selamat datang di Node.js")
$
```

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.

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. 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*

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

readline.js

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

var rl = readline.createInterface({
    input: process.stdin,
      output: process.stdout
});

rl.question("What do you think of node.js? ", function(answer) {
  console.log("Thank you for your valuable feedback:", answer);
  rl.close();
});

// hasil:
// $ node readline.js
// What do you think of node.js? awesome!
// Thank you for your valuable feedback: awesome!
// $
```

function(answer) pada listing di atas merupakan *anonymous function* atau fungsi anonimus (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.

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 mengijinkan pemrogram untuk tidak mendefinisikan tipe data pada saat deklarasi, atau sering juga disebut sebagai *dynamically typed language*:

dynamic.js

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

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

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).

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.

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:

const.js

```
const MENU = "Home";

console.log("Posisi menu = " + MENU);

// mencoba mengisi MENU. berhasil?

MENU = "About";

console.log("Posisi menu = " + MENU);

// Posisi menu = Home
// Posisi menu = Home
```

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).

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 *procedure, routine*, atau *method*.

Definisi Fungsi

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

```
function namaFungsi(argumen1, argumen2, ... , argumentN) {
  ..
  JavaScript code ..
  JavaScript code ..
  JavaScript code ..
  JavaScript code ..
  ..
}
```

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

```
..
..
  namaFungsi(argumen1, argumen2, ..., argumenN);
..
..
```

Contoh dalam program serta pemanggilannya adalah sebagai berikut:

```
$ node
> function addX(angka) {
... console.log(angka + 10);
... }
undefined
> addX(20);
30
undefined
>
> function add2Numbers(angka1, angka2) {
... return angka1 + angka2;
... }
undefined
> console.log("232 + 432 = " + add2Numbers(232, 432));
232 + 432 = 664
undefined
>
```

Fungsi Anonim

Fungsi anonim adalah fungsi tanpa nama, pemrogram tidak perlu memberikan nama ke fungsi. Biasanya fungsi anonimus 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*.

fungsiAnonim.js

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

Fungsi Rekursif

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

fungsiRekursif.js

```
function factorial(n) {

  if ((n == 0) || (n == 1))
    return 1;
  else
    return (n * factorial(n - 1));

}

console.log("factorial(6) = " + factorial(6));

// hasil:
// factorial(6) = 720
```

Fungsi di dalam Fungsi / *Nested Functions*

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

nested.js

```
function induk() {

  var awal = 0;
  function tambahkan() {
    awal++;
  }

  tambahkan();
  tambahkan();

  console.log('Nilai = ' + awal);

}

induk();
tambahkan();

// hasil:
// Nilai = 2
//
// /home/bpdp/kerjaan/git-repos/buku-cloud-nodejs/
//  src/bab-02/nested.js:12
// tambahkan();
// ^
// ReferenceError: tambahkan is not defined
//    at Object.<anonymous> (/home/bpdp/kerjaan/git-repos/
//      buku-cloud-nodejs/src/bab-02/nested.js:12:1)
//    at Module._compile (module.js:456:26)
//    at Object.Module._extensions..js (module.js:474:10)
//    at Module.load (module.js:356:32)
//    at Function.Module._load (module.js:312:12)
//    at Function.Module.runMain (module.js:497:10)
//    at startup (node.js:119:16)
//    at node.js:901:3
```

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.

array.js

```
var arrMembers = ['one','two',,'three',];
// sengaja ada koma di bagian akhir
console.log(arrMembers[0]);
// hasil: one
console.log(arrMembers[2]);
// hasil: undefined
console.log(arrMembers[3]);
// hasil: three
console.log(arrMembers[4]);
// hasil: undefined - karena tidak ada
console.log(arrMembers.length);
// hasil: 4
var multiArray = [
    ['0-0','0-1','0-2'],
    ['1-0','1-1','1-2'],
    ['2-0','2-1','2-2']];
console.log(multiArray[0][2]);
// hasil: 0-2
console.log(multiArray[1][2]);
// hasil: 1-2
```

Literal Boolean

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

boolean.js

```
var isActive = true;

console.log("Current status: " + isActive);

// hasil:
// Current status: true
```

Literal Integer

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

- *binary* (basis 2): digit diawali dengan 0b.
- *decimal* (basis 10): digit tanpa awalan nol.
- *octal* (basis 8): digit diawali dengan 1 angka nol. Pada ECMA-262, bilangan *octal* ini sudah tidak digunakan lagi.

- *hexadecimal* (basis 16): digit diawali dengan 0x.

integers.js

```
var binaryInt = 0b010101000,
    octalInt = 07,
    decimalInt = 34,
    hexInt = 0xAA;

console.log("Nilai inisialisasi");
console.log("==================\n");
console.log("Binary: " + binaryInt);
console.log("Octal: " + octalInt);
console.log("Decimal: " + decimalInt );
console.log("Hex: " + hexInt);
console.log("\nNilai setelah ditambah");
console.log("======================\n");
console.log("Binary + 2: " + (binaryInt + 2));
console.log("Octal + 8: " + (octalInt + 8));
console.log("Decimal + 11: " + (decimalInt + 11));
console.log("Hex + 0xE: " + (hexInt + 0xE));

// Hasil:
// Nilai inisialisasi
// ==================
//
// Binary: 168
// Octal: 7
// Decimal: 34
// Hex: 170
//
// Nilai setelah ditambah
// ======================
//
// Binary + 2: 170
// Octal + 8: 15
// Decimal + 11: 45
// Hex + 0xE: 184
```

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.

floatingPoint.js

```
var fpPertama = 3.1415926,
    fpKedua = -.123456789,
    fpKetiga = -3.1E+12,
    fpKeempat = .1e-20;

console.log("fpPertama: " + fpPertama);
console.log("fpKedua: " + fpKedua);
console.log("fpKetiga: " + fpKetiga);
console.log("fpKeempat: " + fpKeempat);

// Hasil:
// fpPertama: 3.1415926
// fpKedua: -0.123456789
// fpKetiga: -3100000000000
// fpKeempat: 1e-21
```

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

Contoh terakhir di atas menggunakan karakter khusus (**\n**). Karakter khusus ini sering disebut dengan *escape character*. 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.

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 berikut (diambil dari http://en.wikipedia.org/wiki/JSON dengan sedikit perubahan):

json.js

```
var data = {
    "firstName": "John",
    "lastName": "Smith",
    "age": 25,
    "address": {
        "streetAddress": "21 2nd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": "10021"
    },
    "phoneNumber":
    {
        "home": "212 555-1234",
        "fax": "646 555-4567"
    }
}

console.log(data.firstName + " " + data.lastName +
        " has this phone number = "
        + data.phoneNumber.home );

// hasil:
// John Smith has this phone number = 212 555-1234
```

Dari representasi di atas, kita bisa membaca:

- Nilai data *firstname* adalah *John*
- Data *address* terdiri atas sub data *streetAddress*, *city*, *state*, dan *postalCode* yang masing-masing

mempunyai nilai data sendiri-sendiri.

- dan seterusnya

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.

if.js

```
var kondisi = false;
if (kondisi) {
    console.log('hanya dikerjakan jika kondisi bernilai benar/true');
};
// hasil: n/a, tidak ada hasilnya
var kondisi = true;
if (kondisi) {
    console.log('hanya dikerjakan jika kondisi bernilai benar/true');
};
// hasil: hanya dikerjakan jika kondisi bernilai benar/true
// Contoh berikut lebih kompleks, melibatkan input

var readline = require('readline');

var rl = readline.createInterface({
    input: process.stdin,
      output: process.stdout
});

rl.question("Masukkan angka nilai: ", function(answer) {
  if (answer > 80) {
    console.log("Nilai: A");
  } else if (answer > 70) {
    console.log("Nilai: B");
  } else if (answer > 40) {
    console.log("Nilai: C");
  } else if (answer > 30) {
    console.log("Nilai: D");
  } else {
    console.log("Tidak lulus");
  }
  rl.close();
});

// hasil:
// hanya dikerjakan jika kondisi bernilai benar/true
// Masukkan angka nilai: 50
// Nilai: C
```

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.

switch.js

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

var rl = readline.createInterface({
    input: process.stdin,
      output: process.stdout
});

console.log("Menu");
console.log("====");
console.log("1. Mengisi data");
console.log("2. Mengedit data");
console.log("3. Menghapus data");
console.log("4. Mencari data");
rl.question("Masukkan angka pilihan anda: ", function(answer) {
  console.log("Pilihan anda: " + answer);
  switch (answer) {
    case "1":
      console.log("Anda memilih menu pengisian data");
      break;
    case "2":
      console.log("Anda memilih menu pengeditan data");
      break;
    case "3":
      console.log("Anda memilih menu penghapusan data");
      break;
    case "4":
      console.log("Anda memilih menu pencarian data");
      break;
    default:
      console.log("Anda tidak memilih salah satu dari menu di atas");
      break;
  }
  rl.close();
});

// hasil:
// $ node switch.js
// Menu
// ====
// 1. Mengisi data
// 2. Mengedit data
// 3. Menghapus data
// 4. Mencari data
// Masukkan angka pilihan anda: 10
// Pilihan anda: 10
// Anda tidak memilih salah satu dari menu di atas
```

```
// $ node switch.js
// Menu
// ====
// 1. Mengisi data
// 2. Mengedit data
// 3. Menghapus data
// 4. Mencari data
// Masukkan angka pilihan anda: 2
// Pilihan anda: 2
// Anda memilih menu pengeditan data
```

Looping

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

for

for.js

```
for (var i = 0; i < 9; i++) {
  console.log(i);
}

// hasil:
// 0
// 1
// 2
// 3
// 4
// 5
// 6
// 7
// 8
```

Pernyataan ``for'' juga bisa digunakan untuk mengakses data yang tersimpam dalam struktur data JavaScript (JSON).

forIn.js

```
var data = {a:1, b:2, c:3};

for (var iterasi in data) {
  console.log("Nilai dari iterasi " + iterasi + " adalah: " + data[iterasi]);
}

// hasil:
// Nilai dari iterasi a adalah: 1
// Nilai dari iterasi b adalah: 2
// Nilai dari iterasi c adalah: 3
```

do .. while

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

doWhile.js

```
var i = 0;
do {
  i += 2;
  console.log(i);
} while (i < 20);

// hasil:
// 2
// 4
// 6
// 8
// 10
// 12
// 14
// 16
// 18
// 20
```

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 expresi dievaluasi sudah bernilai *false*.

while.js

```
var n = 0;
var x = 0;

while (n < 5) {
  n ++;
  x += n;
  console.log("Nilai n = " + n);
  console.log("Nilai x = " + x);
}

// hasil:
// Nilai n = 1
// Nilai x = 1
// Nilai n = 2
// Nilai x = 3
// Nilai n = 3
// Nilai x = 6
// Nilai n = 4
// Nilai x = 10
// Nilai n = 5
// Nilai x = 15
```

label, break, dan continue

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

- *label* digunakan untuk memberi pengenal pada suatu lokasi program sehingga bisa direferensi oleh *break* maupun *continue* (jikaa 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.

breakContinue.js

```
var n = 0;
var x = 0;

while (n < 5) {
  n ++;
  x += n;

  if (x%2 == 0) {
    continue;
  };

  if (x>10) {
    break;
  };

  console.log("Nilai n = " + n);
  console.log("Nilai x = " + x);

};
// hasil:
//Nilai n = 1
//Nilai x = 1
//Nilai n = 2
//Nilai x = 3
```

breakWithLabel.js

```
topLabel:
  for(var k = 0; k < 10; k++){
    for(var m = 0; m < 20; m++){
      if(m == 5){
        console.log("Nilai k = " + k);
        console.log("Nilai m = " + m);
        break topLabel;
      }
    }
  }
// hasil:
//Nilai k = 0
//Nilai m = 5
```

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

try.js

```
try {
  gakAdaFungsiIni();
} catch (e) {
  console.log ("Error: " + e.message);
} finally {
  console.log ("Bagian 'pembersihan', akan dikerjakan, apapun yang terjadi");
};

// hasil:
// Error: gakAdaFungsiIni is not defined
// Bagian 'pembersihan', akan dikerjakan, apapun yang terjadi
```

throw.js

```
try {
  var a = 1/0;
  throw "Pembagian oleh angka 0";
} catch (e) {
  console.log ("Error: " + e);
};

// hasil:
// Error: Pembagian oleh angka 0
```

Paradigma Pemrograman di JavaScript

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, Ocaml, Haskell, F#, 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 di-evaluasi 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. Bab ini akan membahas beberapa fitur pemrograman fungsional di JavaScript. Pembahasan ini didasari pembahasan di bab sebelumnya tentang Fungsi di JavaScript.

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 anonimus (fungsi yang dipanggil/dieksekusi tanpa ditautkan (*bound*) ke suatu *identifier*). Berikut adalah implementasi dari konsep ini di JavaSCript:

lambda.js

```
// Diambil dari
// http://stackoverflow.com/questions/3865335/what-is-a-lambda-language
// dengan beberapa perubahan

function applyOperation(a, b, operation) {
  return operation(a, b);
}

function add(a, b) {
  return a+b;
}

function subtract(a, b) {
  return a-b;
}

console.log('1,2, add: ' + applyOperation(1,2, add));
console.log('43,21, subtract: ' + applyOperation(43,21, subtract));

console.log('4^3: ' + applyOperation(4, 3, function(a,b) {return Math.pow(a, b)}))

// hasil:
// 1,2, add: 3
// 43,21, subtract: 22
// 4^3: 64
```

Higher-order Function

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

hof.js

```
function forEach(array, action) {
  for (var i = 0; i < array.length; i++ )
    action(array[i]);
}

function print(word) {
  console.log(word);
}

function makeUpperCase(word) {
  console.log(word.toUpperCase());
}

forEach(["satu", "dua", "tiga"], print);
forEach(["satu", "dua", "tiga"], makeUpperCase);

// hasil:
//satu
//dua
//tiga
//SATU
//DUA
//TIGA
```

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.

closure.js

```
// Diambil dengan sedikit perubahan dari:
// https://developer.mozilla.org/en-US/docs/JavaScript/Guide/Closures
function makeAdder(x) {
  return function(y) {
    return x + y;
  };
}

var add5 = makeAdder(5);
var add10 = makeAdder(10);

console.log(add5(2));  // 7
console.log(add10(2)); // 12
```

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 tersediaa tersebut.

currying.js

```
// Diambil dari:
// http://javascriptweblog.wordpress.com/2010/04/05/
//      curry-cooking-up-tastier-functions/
// dengan sedikit perubahan

function toArray(fromEnum) {
    return Array.prototype.slice.call(fromEnum);
}

Function.prototype.curry = function() {
    if (arguments.length<1) {
        return this; //nothing to curry with - return function
    }
    var __method = this;
    var args = toArray(arguments);
    return function() {
        return __method.apply(this, args.concat(toArray(arguments)));
    }
}

var add = function(a,b) {
    return a + b;
}

//create function that returns 10 + argument
var addTen = add.curry(10);
console.log(addTen(20)); //30
```

Pemrograman Berorientasi Obyek

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 yng 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*.

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 method namaMethod pada kelas Kelas. Perhatikan contoh pada listing berikut.

obyek.js

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

// Definisi obyek
function Halaman(alamatUrl) {
  this.url = alamatUrl;
  console.log("Mengakses alamat " + alamatUrl);
}

Halaman.prototype.getDomainName = function() {
  return url.parse(this.url, true).host;
}
// sampai disini definisi obyek
// Halaman.prototype.getDomainName => menetapkan method getDomainName
// untuk obyek

var halSatu = new Halaman("http://nodejs.org/api/http.html");
var halDua = new Halaman("http://bpdp.name/login?fromHome");

console.log("Alamat URL yang diakses oleh halSatu = " + halSatu.url);
console.log("Alamat URL yang diakses oleh halDua = " + halDua.url);

console.log("Nama domain halDua = " + halDua.getDomainName());

// hasil:
// Mengakses alamat http://nodejs.org/api/http.html
// Mengakses alamat http://bpdp.name/login?fromHome
// Alamat URL yang diakses oleh halSatu = http://nodejs.org/api/http.html
// Alamat URL yang diakses oleh halDua = http://bpdp.name/login?fromHome
// Nama domain halDua = bpdp.name
```

Inheritance / **Pewarisan**

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

inheritance.js

```
// Definisi obyek
function Kelas(param) {
  this.property1 = new String(param);
}

Kelas.prototype.methodSatu = function() {
  return this.property1;
}

var kelasSatu = new Kelas("ini parameter 1 dari kelas 1");

console.log("Property 1 dari kelasSatu = " + kelasSatu.property1);
console.log("Property 1 dari kelasSatu, diambil dari method  = " + kelasSatu.methodSatu
());

// Definisi inheritance:
// SubKelas merupakan anak dari Kelas yang didefinisikan
// di atas.

SubKelas.prototype = new Kelas();
SubKelas.prototype.constructor = SubKelas;

function SubKelas(param) {
  this.property1 = new String(param);
}

// method overriding
SubKelas.prototype.methodSatu = function(keHurufBesar) {
  console.log("Ubah ke huruf besar? = " + keHurufBesar);
  if (keHurufBesar) {
    return this.property1.toUpperCase();
  } else {
    return this.property1.toLowerCase();
  }
}

SubKelas.prototype.methodDua = function() {
  console.log("Berada di method dua dari SubKelas");
}

// mari diuji
```

```
var subKelasSatu = new SubKelas("Parameter 1 Dari Sub Kelas 1");

console.log("Property 1 dari sub kelas 1 = " + subKelasSatu.property1);
console.log("Property 1 dari sub kelas 1, dr method+param = " + subKelasSatu.methodSatu
(true));
console.log("Property 1 dari sub kelas 1, dr method+param = " + subKelasSatu.methodSatu
(false));

console.log(subKelasSatu.methodDua());
// hasil:
//
//Property 1 dari kelasSatu = ini parameter 1 dari kelas 1
//Property 1 dari kelasSatu, diambil dari method  = ini
//parameter 1 dari kelas 1
//Property 1 dari sub kelas 1 = Parameter 1 Dari Sub Kelas 1
//Ubah ke huruf besar? = true
//Property 1 dari sub kelas 1, dr method+param =
//PARAMETER 1 DARI SUB KELAS 1
//Ubah ke huruf besar? = false
//Property 1 dari sub kelas 1, dr method+param =
//parameter 1 dari sub kelas 1
//Berada di method dua dari SubKelas
```

Mengelola Paket Menggunakan npm

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://www.npmjs.com. Node.js menyediakan perintah *npm* untuk mengelola paket pustaka di repositori tersebut. Untuk menggunakan utilitas ini,pemrogram harus terkoneksi dengan Internet.

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 mempunya 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 berikut:

```
Usage: npm <command>

where <command> is one of:
    access, add-user, adduser, apihelp, author, bin, bugs, c,
    cache, completion, config, ddp, dedupe, deprecate, dist-tag,
    dist-tags, docs, edit, explore, faq, find, find-dupes, get,
    help, help-search, home, i, info, init, install, issues, la,
    link, list, ll, ln, login, logout, ls, outdated, owner,
    pack, ping, prefix, prune, publish, r, rb, rebuild, remove,
    repo, restart, rm, root, run-script, s, se, search, set,
    show, shrinkwrap, star, stars, start, stop, t, tag, team,
    test, tst, un, uninstall, unlink, unpublish, unstar, up,
    update, upgrade, v, verison, version, view, whoami

npm <cmd> -h     quick help on <cmd>
npm -l           display full usage info
npm faq          commonly asked questions
npm help <term>  search for help on <term>
npm help npm     involved overview

Specify configs in the ini-formatted file:
    /home/bpdp/.npmrc
or on the command line via: npm <command> --key value
Config info can be viewed via: npm help config

npm@2.14.7 /opt/software/node-v4.2.1-linux-x64/lib/node_modules/npm
```

Lokasi direktori pada sintaks penggunaan *npm* tersebut adalah lokasi spesifik di komputer penulis (misalnya: **/home/bpdp/.npmrc**).

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

Instalasi Paket

npm sebenarnya bukan merupakan singkatan dari *Node Package Manager*, meskipun seringkali orang menterjemahkan dengan singkatan tersebut. *npm* seharusnya ditulis dalam huruf kecil semua seperti yang dijelaskan pada FAQ. *npm* merupakan perintah di *shell / command line*. Struktur perintahnya adalah: *npm perintah argumen*. Instalasi paket dilakukan dengan perintah berikut:

```
$ npm install namapaket
```

Perintah diatas akan memasang versi terakhir dari paket *namapaket*.

Struktur Instalasi Paket Node.js

Dalam installasi paket pustaka, berkas-berkas akan terletak dalam folder lokal aplikasi *node_modules*. Pada mode installasi paket pustaka global (dengan -g atau –global dibelakang baris perintah), paket pustaka akan dipasang pada */usr/lib/node_modules* (dengan lokasi installasi 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.

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:

```
$ npm ls
```

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.

```
$ npm ls --depth=0
weberiajs@1.0.0 /home/bpdp/kerjaan/git-repos/weberia/weberiajs
 ├──── UNMET DEPENDENCY hapi@^11.0.2
 └──── jsonld@0.4.2

npm ERR! missing: hapi@^11.0.2, required by weberiajs@1.0.0
```

Jika sudah terinstall, perintah "npm ls" akan menampilkan struktur dari paket yang telah terinstall dalam bentuk struktur pohon.

Menghapus Paket / *Uninstall*

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

```
$ npm uninstall namapaket
```

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':

```
$ npm search sha512
```

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

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 *arangojs*:

```
$ npm info arangojs
```

Memperbaharui Paket

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

```
$ npm update
```

Node.js dan Web: Teknik Pengembangan Aplikasi

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.

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

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 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.

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 menangania 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.

Mengakses Basis Data NoSQL: mongoDB

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'' [5: http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/nosql/Home%20Page]. 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.

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:

- 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

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):

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

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 [fig:mongowebadminconsole]. Sementara itu, untuk mengakses server menggunakan shell, bisa digunakan perintah *mongo* sebagai berikut:

[Admin web console untuk mongoDB]

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*.

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

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:

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

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**.

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

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:

Setelah itu, buat program sesuai dengan listing program berikut.

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. 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*.

Pola Arsitektur Aplikasi Web: MVC dan ExpressJS

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.

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 [fig:mvc]

[Pola arsitektur MVC]

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

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.

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:

Struktur direktori tersebut akan diubah sesuai dengan pola MVC:

Beberapa perubahan terhadap struktur direktori:

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

File-file yang Diperlukan

Beberapa file diubah isinya dan ada juga file yang baru.

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 [fig:hasil-mvc]

[Pola arsitektur MVC]

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

Real-time Web Menggunakan Socket.io

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.

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.

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 XMLa tetapi JSON).
- Obyek XMLHttpRequest untuk komunikasi asynchronous
- JavaScript

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

WebSocket

WebSocket merupakan teknologi Web yang menyediakan saluran komunikasi full duplex pada satu koneksi TCP. Protokol WebSocket distandarkan oleh IETF di RFC 6455 sedangkap 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/

Socket.io

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)

Menggunakan Socket.io untuk Real-time Web

Socket.io melibatkan sisi klien dan sisi server. Pada sisi server, paket yang diperlukan adalah cocket.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:

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*:

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.

Menjalankan Server Socket.io

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

```
<!DOCTYPE html>
<html>
    <head>
        <title>Contoh Socket.io + Express</title>
        <link rel="stylesheet" href="/stylesheets/style.css">
    </head>
    <body>
        <h1>Contoh Socket.io + Express</h1>
        <p>Contoh Socket.io + Express</p>
        <script src="/javascripts/socket.io.js"></script>
        <script>
            var socket = io.connect('http://localhost');
            socket.on('kirim ke browser', function (data) {
              document.getElementById("container").innerHTML=
                "<p>" + data.kalimatDariServer + "</p>";
            socket.emit('dari browser', {
            kalimatDariBrowser: 'Kalimat ini dikirim dari browser' });
            });
        </script>
        <div id="container">
            <p>Contoh Socket.io + Express</p>
        </div>
    </body>
</html>
```

Tampilan di browser bisa dilihat pada gambar [fig:socket-io-express] [Hasil di browser dari ExpressJS Socket.io] Contoh pada materi ini merupakan contoh sederhana, tetapi diharapkan bisa dengan mudah dipahami untuk membuat aplikasi Web real-time.

Daftar Pustaka

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