

# **Laporan Ujian Akhir Semester Pengembangan Aplikasi Terdistribusi**

**IF4031 – Pengembangan Aplikasi Terdistribusi  
Semester I Tahun 2014 / 2015**



oleh :

Iskandar Setiadi / 13511073

Sekolah Teknik Elektro dan Informatika (STEI ITB)

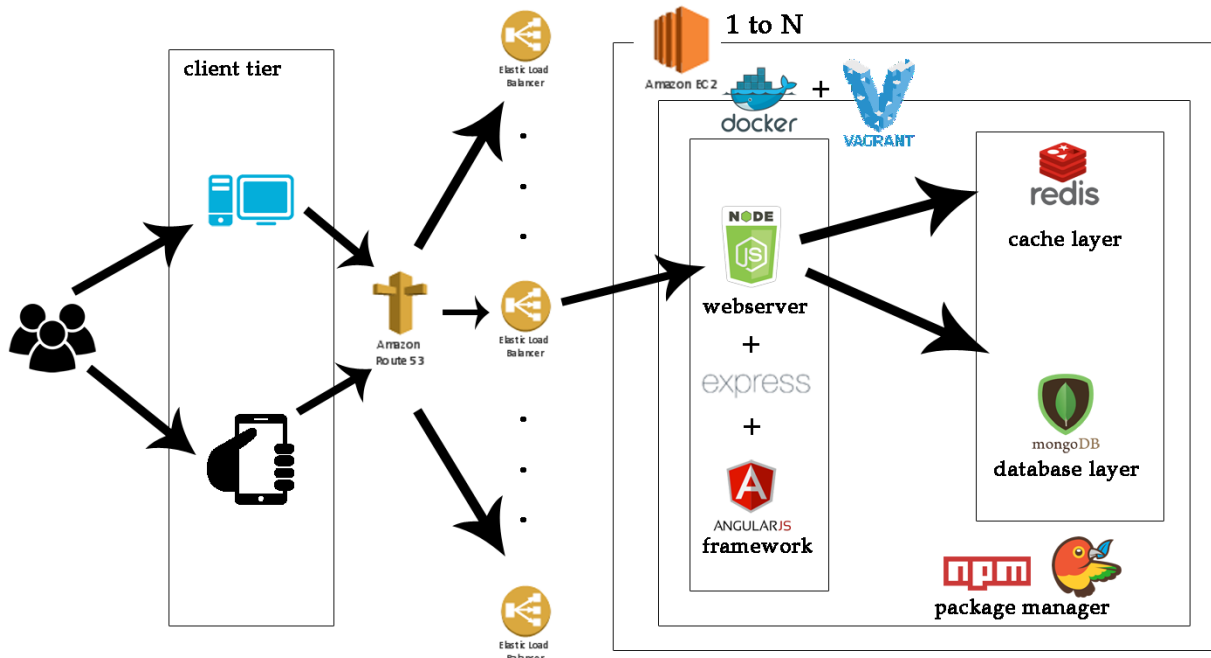
Institut Teknologi Bandung

Jl. Ganesha No. 10, Bandung 40132

Tahun 2014

## I. Usulan arsitektur aplikasi & pertimbangan

Arsitektur aplikasi yang akan digunakan untuk sistem *ticketing* pesawat adalah sebagai berikut:



Teknologi-teknologi yang digunakan adalah sebagai berikut:

1. Amazon Route 53 sebagai *DNS web service*
2. Amazon Elastic Load Balancer sebagai *load balancer*
3. Amazon EC2 sebagai *hosting dari web service*
4. Docker sebagai *container dari aplikasi*
5. Vagrant sebagai *virtual development environments*
6. NodeJS sebagai *web server*
7. ExpressJS sebagai *framework*
8. AngularJS sebagai *framework*
9. Redis sebagai *cache layer*
10. MongoDB (+Mongoose) sebagai *database layer*
11. NPM sebagai *javascript management*
12. Bower sebagai *package management*

Arsitektur ini digunakan untuk memberikan sistem *web-based* yang memiliki performansi cepat, *scalability* tinggi, *reliability* tinggi, dan mendukung *extensibility* yang baik.

1. Pengguna dapat melakukan pemesanan tiket/booking hotel melalui aplikasi web

➔ Untuk menyelesaikan permasalahan ini, halaman web dibangun diatas **Node JS**. Node JS memiliki performansi yang cukup cepat karena sifatnya yang merupakan *event-driven* serta *asynchronous (non-blocking I/O)*. Selain itu, penggunaan *framework* seperti **Express JS (back-end)** + **Angular JS (front-end)** mempermudah tahap pengembangan untuk versi *web browser* pada *desktop* maupun *mobile*.

Apabila aplikasi ini ingin dikembangkan untuk mendukung *desktop* maupun *mobile* pada *client tier*, Angular JS memiliki *Material Design* yang dapat diakses di:

<https://material.angularjs.org/>.

2. Untuk melakukan reservasi atau melihat ketersediaan tiket, aplikasi harus terhubung ke aplikasi eksternal menggunakan web service API yang mungkin memerlukan waktu lama (dapat mencapai hingga 1-3 menit)

➔ Untuk menyelesaikan permasalahan ini, sistem ini memiliki *cache layer* yang diimplementasikan dengan **Redis** serta *database layer* yang diimplementasikan dengan **MongoDB** (+Mongoose sebagai *library* antara MongoDB dengan Node).

Redis (*cache*) digunakan untuk menyimpan ketersediaan tiket dikarenakan informasi ketersediaan tiket akan sering diakses oleh pengguna dan tidak sering berubah dalam rentang waktu 1 – 5 menit. Oleh karena itu, sistem dapat melakukan penarikan data secara *asynchronous* dari aplikasi eksternal dan menyimpan informasi ketersediaan tiket dalam *cache layer*. Redis memiliki operasi “EXPIRE key seconds” yang dapat digunakan untuk mengeset batasan waktu validitas suatu data.

Dokumentasi: <http://redis.io/commands/expire>

Dengan adanya *cache layer*, operasi pembacaan ketersediaan tiket yang akan jauh lebih sering diakses dibandingkan reservasi tiket dapat dilayani dengan cepat.

Untuk melakukan *booking* / reservasi tiket, sistem dapat melayani pengguna secara *asynchronous (non-blocking)* dengan mudah melalui arsitektur Node JS.

MongoDB digunakan untuk menyimpan data pengguna seperti *username*, *password*, dll. MongoDB memiliki konsistensi yang baik dan *availability* yang cukup baik untuk sistem terdistribusi.

3. Sistem harus scalable, mampu meningkatkan kapasitas transaksi dengan menambah *resources* (komputer) tanpa memerlukan rekonfigurasi ulang aplikasi yang besar.

➔ Untuk menjamin skalabilitas, sistem ini dibangun diatas *container Docker* dan **Vagrant**. Docker mendukung pengisolasian kode dengan *library* dependensinya, sehingga migrasi dapat dilakukan dengan mudah. Selain itu, Vagrant memberikan dukungan tambahan dalam melakukan konfigurasi lingkungan *virtual* untuk pengembangan aplikasi.

Untuk *deployment*, aplikasi ini sudah dibangun diatas **Amazon EC2**. Apabila dibutuhkan *scaling* ke jenis *instance* yang lebih tinggi (misalnya t1.micro ke c3.large), dapat dilakukan dengan meng-*export* VM *image* melalui tutorial:

<http://aws.amazon.com/ec2/vm-import/>

Apabila pengembang ingin melakukan migrasi ke VPS lain seperti DigitalOcean dsb, Docker + Vagrant telah memberikan enkapsulasi yang mempermudah skalabilitas maupun migrasi aplikasi.

Selain itu, basis data yang menggunakan MongoDB telah mendukung proses *replication* (*vertical scaling*) maupun *sharding* (*horizontal scaling*), sehingga sistem ini mendukung skalabilitas yang baik.

4. Sistem harus reliable, mampu tetap operasional meskipun ada satu atau lebih komputer yang down

➔ Untuk menjamin reliabilitas, sistem ini menggunakan infrastruktur **AWS Route 53** dan **AWS Elastic Load Balancer**. Route 53 digunakan sebagai DNS *mapper*, sehingga sistem dapat melakukan proses *redirect* ke lebih dari satu *load balancer*. Setiap *load balancer* dapat menangani 1 sampai N buah *virtual server* AWS EC2. Apabila terdapat satu atau lebih *instance* EC2 yang mati, Elastic Load Balancer dapat menyeimbangkan *load* ke *server* lain yang masih tersedia. Sedangkan apabila

terdapat gangguan pada Elastic Load Balancer, AWS Route 53 akan menangani proses *mapping* dari DNS ke alamat *load balancer* yang lain.

5. Sistem harus *extensible*, dapat dikembangkan untuk menyediakan fungsionalitas baru, misal untuk layanan B2B atau via web service API untuk melakukan reservasi dari perusahaan lain, aplikasi desktop/mobile untuk melakukan reservasi

➔ Penggunaan *framework* seperti **Express JS** dan **Angular JS** serta penggunaan dependensi *manager* seperti **npm** dan **bower** telah menjamin ekstensibilitas dari sistem. Pembangunan kode yang modular (*MVC design pattern*) dan dependensi *library* yang modular (*package.json* dan *bower.json*) menjamin penambahan modul maupun penghapusan modul dapat dilakukan dengan mudah.

Selain itu, protokol API yang digunakan adalah RESTful. Untuk pengembangan kedepannya, kita dapat mengenkapsulasi *web service* ini menggunakan satu modul dengan struktur *error code*, jenis *call*, dll yang sudah didefinisikan sebelumnya.

Alternatif lainnya, *web service* API ini dapat dikembangkan lebih lanjut dengan WSO2 API Manager apabila jumlah API telah mencapai tahap yang sulit untuk *maintenance* secara manual. (Opsional)

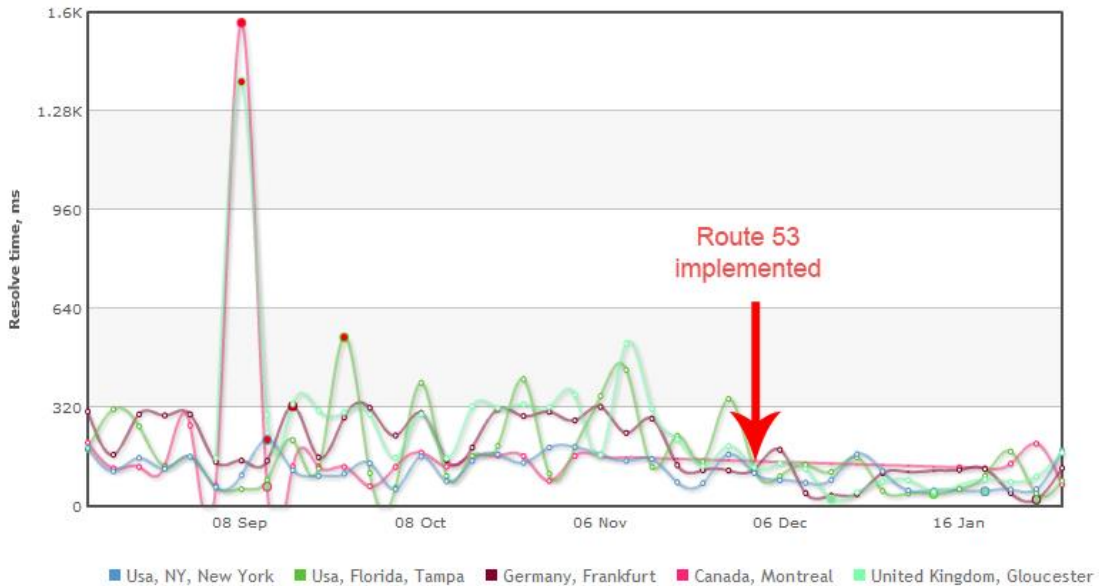
## II. Teknologi yang digunakan & pertimbangan

- Amazon Route 53 sebagai DNS *web service*, Amazon Elastic Load Balancer sebagai *load balancer*, dan Amazon EC2 sebagai *hosting* dari *web service*

Salah satu alasan utama penggunaan Amazon sebagai *backbone* dari infrastruktur sistem terdistribusi adalah skalabilitas dan ketersediaan (*availability*) dari layanan yang sudah terjamin. Penggunaan Route 53 memiliki beberapa kegunaan, seperti mendukung penggunaan lebih dari satu *load balancer* serta mengurangi waktu *resolve time* dari alamat *server*. Gambar berikut ini merupakan salah satu hasil *benchmark*<sup>1</sup> yang menunjukkan performansi Route 53 jika dibandingkan dengan DNS *resolver* biasa.

---

<sup>1</sup> Referensi: <http://www.x-pose.org/2011/02/amazon-route-53-benchmark-comparisons/>

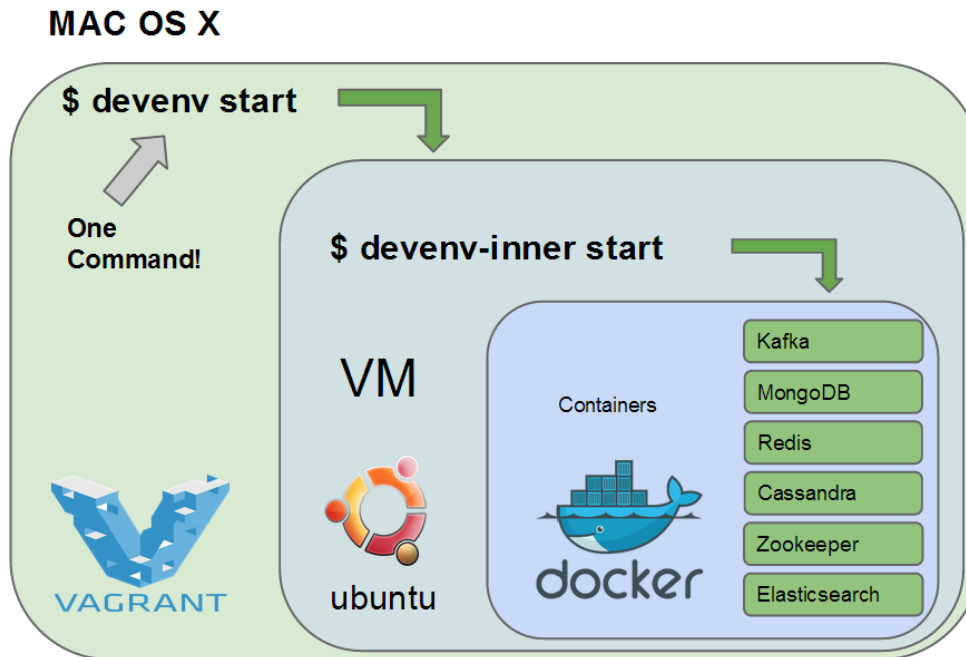


Penggunaan Amazon Load Balancer untuk Amazon EC2 dapat meningkatkan reliabilitas sistem secara keseluruhan. Hal ini dikarenakan *data center* Amazon yang tersebar di berbagai *region*, sehingga memberikan jaminan *availability* dan *reliability* yang tinggi. Amazon EC2 memiliki skalabilitas yang baik karena proses migrasi maupun *upgrade server* dapat dilakukan dengan mudah. *Pay-as-you-go* merupakan salah satu keuntungan dalam menggunakan *cloud infrastructure* karena kita dapat dengan mudah mengganti tipe *instance* sesuai kebutuhan saat tersebut. Misalnya, saat *peak time*, kita dapat menambah jumlah *server* dan berlaku sebaliknya.

- Docker sebagai *container* dari aplikasi dan Vagrant sebagai *virtual development environments*

Docker dan Vagrant dapat digunakan untuk enkapsulasi aplikasi. Gambar berikut ini<sup>2</sup> menunjukkan gambaran umum terhadap cara kerja Docker dan Vagrant sebagai *container* aplikasi dan *virtual development environments*. Dengan dua aplikasi ini, pengembangan sistem dapat dilakukan di tempat yang berbeda dan dikerjakan oleh banyak orang tanpa mengalami kesulitan dalam mengatur dependensi *library* maupun konfigurasi lainnya.

<sup>2</sup> Referensi: <https://blog.relateiq.com/a-docker-dev-environment-in-24-hours-part-2-of-2/>

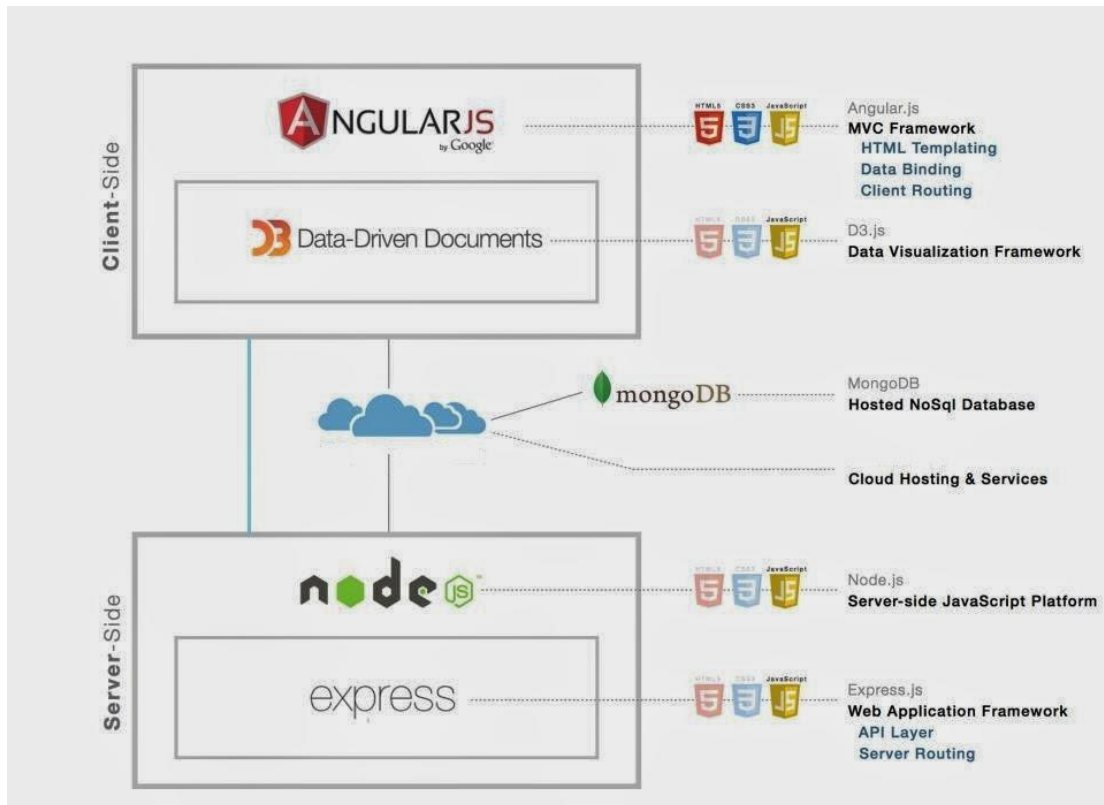


Selain itu, Docker dan Vagrant memberikan kemudahan apabila dibutuhkan migrasi dari infrastruktur *cloud* AWS ke infrastruktur lainnya.

- NodeJS sebagai *web server*, ExpressJS sebagai *backend framework*, AngularJS sebagai *frontend framework*, dan MongoDB (+Mongoose) sebagai *database layer*

NodeJS merupakan salah satu *web server* yang digunakan untuk menjalankan halaman web berbasis bahasa Javascript. Keuntungan utama penggunaan NodeJS adalah sifat bahasanya yang *non-blocking*, sehingga mudah digunakan untuk memberikan performansi yang baik dalam permasalahan *asynchronous*. Selain itu, penggunaan *framework* seperti ExpressJS dan AngularJS meningkatkan ekstensibilitas dari sistem. Hal ini dikarenakan struktur kode dari aplikasi kita menjadi modular (terbagi berdasarkan *design pattern* serta *frontend-backend*). Gambar dibawah ini<sup>3</sup> merupakan salah satu gabungan aplikasi Node yang dianggap sebagai *best practice*, yaitu MEAN (Mongo + Express + Angular + Node JS).

<sup>3</sup> Referensi: [http://blog.harriersys.com/2014/05/web-application-development-becomes\\_31.html](http://blog.harriersys.com/2014/05/web-application-development-becomes_31.html)



Penggunaan MongoDB sebagai *database layer* menjadi salah satu pilihan utama karena MongoDB memiliki fitur yang cukup *robust*. MongoDB dapat menjamin *availability* yang tinggi, memiliki fitur replikasi yang cukup baik, serta mendukung *sharding* (*horizontal scaling*) dari sebuah basis data NoSQL. Selain itu, MongoDB cukup mudah untuk di *deploy* serta memiliki *support* yang baik dalam Node JS maupun Docker.

#### - Redis sebagai *cache layer*

Untuk data yang cukup lama untuk didapatkan (*external web API*) dan data tersebut tidaklah kritikal, kita dapat menggunakan *cache layer* untuk mempercepat *response time* terhadap pengguna. Redis menjadi pilihan utama sistem ini dikarenakan Redis memiliki performansi yang sangat cepat dan mudah untuk dikonfigurasi. Salah satu perusahaan yang menggunakan Redis sebagai *cache layer* adalah Stack Overflow<sup>4</sup>. Setiap data yang disimpan dalam *cache layer* memiliki waktu kadaluarsa / *expire*, dan Redis memiliki *support* yang baik terhadap permasalahan tersebut.

<sup>4</sup> Referensi: <http://meta.stackexchange.com/questions/69164/does-stack-overflow-use-caching-and-if-so-how>

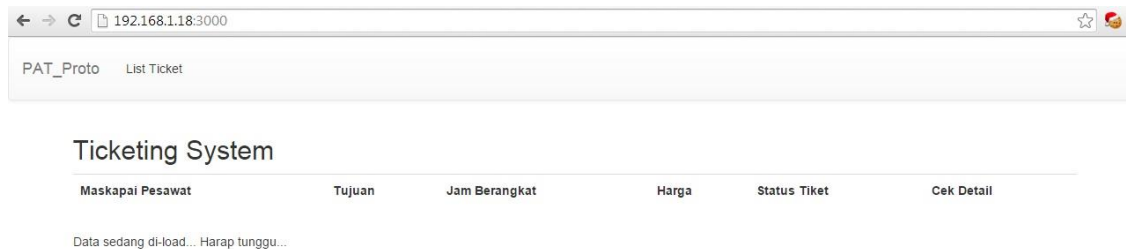


- NPM sebagai *javascript management* dan bower sebagai *package management*

Aplikasi yang berskala besar akan sangat sulit untuk dikembangkan tanpa *package management*. Penggunaan *package management* dapat meningkatkan ekstensibilitas dari sistem. Beberapa contoh aplikasi untuk *package management* adalah npm, bower (*javascript*), composer (PHP), dan maven (Java). Aplikasi / *tools* ini digunakan untuk membantu proses instalasi dependensi sampai proses *build* aplikasi secara keseluruhan. *Tools* ini berguna untuk digunakan terutama dalam mendukung *project* yang melibatkan banyak orang dan perubahan aplikasi yang terjadi dengan cepat. NPM merupakan salah satu *management* bawaan dari Node JS (*back-end*) dan bower memberikan dukungan *package management* untuk *front-end*.

### III. Screenshot hasil

Ketika sistem pertama kali dinyalakan, maka *cache layer* dari sistem tersebut masih kosong. Oleh karena itu, beberapa *request* pertama dari pengguna memiliki *response time* yang sangat lambat karena data mengenai informasi tiket belum ditarik eksternal API. Untuk menghindari halaman *web* yang tidak responsif, penggunaan *non-blocking I/O* dengan Node JS akan menghasilkan tampilan sebagai berikut:



Disisi *server*, *cache layer* akan melakukan pemanggilan ke *web service* eksternal dalam periode waktu tertentu (misalnya 50 detik). Hal ini dilakukan untuk mencegah *cache layer* dalam keadaan kosong dan menyebabkan pengguna menunggu lama. Cara ini akan

efisien jika jumlah pengguna yang mengakses *server* tersebut banyak (*high traffic*). Tampilan berikut ini (dari sisi *server*) menunjukkan bahwa isi *cache layer* masih kosong:

```
^Cfreedomofkeima@freedomofkeima:~/Desktop/pat-project$ node app.js
body-parser deprecated bodyParser: use individual json/urlencoded middlewares app.js:30:9
body-parser deprecated undefined extended: provide extended option node_modules/body-parser/index.js:85:29
Express server listening on port 3000
checkKey: null
GET /favicon.ico 304 2266.786 ms - -
[{"_id":"54969bb0b3de2200004185ac","id":"GAN","avail":"2014-12-11T18:12:44.000Z","tujuan":"bandung","harga":"500"}, {"_id":"5494c511e3f6464226c0f1dc","id":"AFR","avail":"2014-12-11T18:12:44.000Z","tujuan":"malaysia","harga":"200"}, {"_id":"5494c4d9e3f6464226c0f1db","id":"BOE","avail":"2014-12-11T09:23:46.000Z","tujuan":"singapore","harga":"300"}, {"_id":"5494c450e3f6464226c0f1da","id":"BOE","avail":"2014-12-10T08:14:55.000Z","tujuan":"singapore","harga":"300"}]
54969bb0b3de2200004185ac_id timeout in 60 seconds.
54969bb0b3de2200004185ac_id timeout in 60 seconds.
54969bb0b3de2200004185ac_tujuan timeout in 60 seconds.
54969bb0b3de2200004185ac_avail timeout in 60 seconds.
54969bb0b3de2200004185ac_harga timeout in 60 seconds.
5494c511e3f6464226c0f1dc_id timeout in 60 seconds.
5494c511e3f6464226c0f1dc_id timeout in 60 seconds.
5494c511e3f6464226c0f1dc_tujuan timeout in 60 seconds.
5494c511e3f6464226c0f1dc_avail timeout in 60 seconds.
5494c511e3f6464226c0f1dc_harga timeout in 60 seconds.
```

Setelah eksternal API mengembalikan hasil ketersediaan tiket, Node JS akan menampilkan hasil pencarian dengan menggunakan *callback* pada Angular JS. Halaman web akan menampilkan tampilan sebagai berikut:

← → ↻ 192.168.1.18:3000 ☆

PAT\_Proto List Ticket

### Ticketing System

Maskapai Pesawat	Tujuan	Jam Berangkat	Harga	Status Tiket	Cek Detail
GAN	bandung	2014-12-11T18:12:44.000Z	500	Tersedia	<button>Cek</button>
AFR	malaysia	2014-12-11T18:12:44.000Z	200	Tersedia	<button>Cek</button>
BOE	singapore	2014-12-11T09:23:46.000Z	300	Tersedia	<button>Cek</button>
BOE	singapore	2014-12-10T08:14:55.000Z	300	Tersedia	<button>Cek</button>

Cache baru saja diperbaharui

Apabila *cache layer* sudah tidak kosong, maka setiap *request* pengguna dapat dilayani dengan cepat dan menghasilkan tampilan sebagai berikut:

Maskapai Pesawat	Tujuan	Jam Berangkat	Harga	Status Tiket	Cek Detail
GAN	bandung	2014-12-11T18:12:44.000Z	500	Tersedia	<a href="#">Cek</a>
AFR	malaysia	2014-12-11T18:12:44.000Z	200	Tersedia	<a href="#">Cek</a>
BOE	singapore	2014-12-11T09:23:46.000Z	300	Tersedia	<a href="#">Cek</a>
BOE	singapore	2014-12-10T08:14:55.000Z	300	Tersedia	<a href="#">Cek</a>

Data ini diambil dari Cache

Untuk proses *booking*, *cache layer* tidak digunakan karena proses pemesanan membutuhkan data terbaru untuk menjamin konsistensi data. Untuk memberikan *user experience* yang baik bagi pengguna, kita dapat menggunakan *progress bar* atau *loader* untuk memberikan jaminan bahwa web masih responsif (tidak *hang*). Berikut ini adalah tampilan halaman pemesanan tiket:

Informasi Pemesanan

Kode Tiket : 54969bb0b3de2200004185ac

Status Tiket : Tersedia

Informasi Pembeli

Nama Pemesan:

Iskandar Setiadi

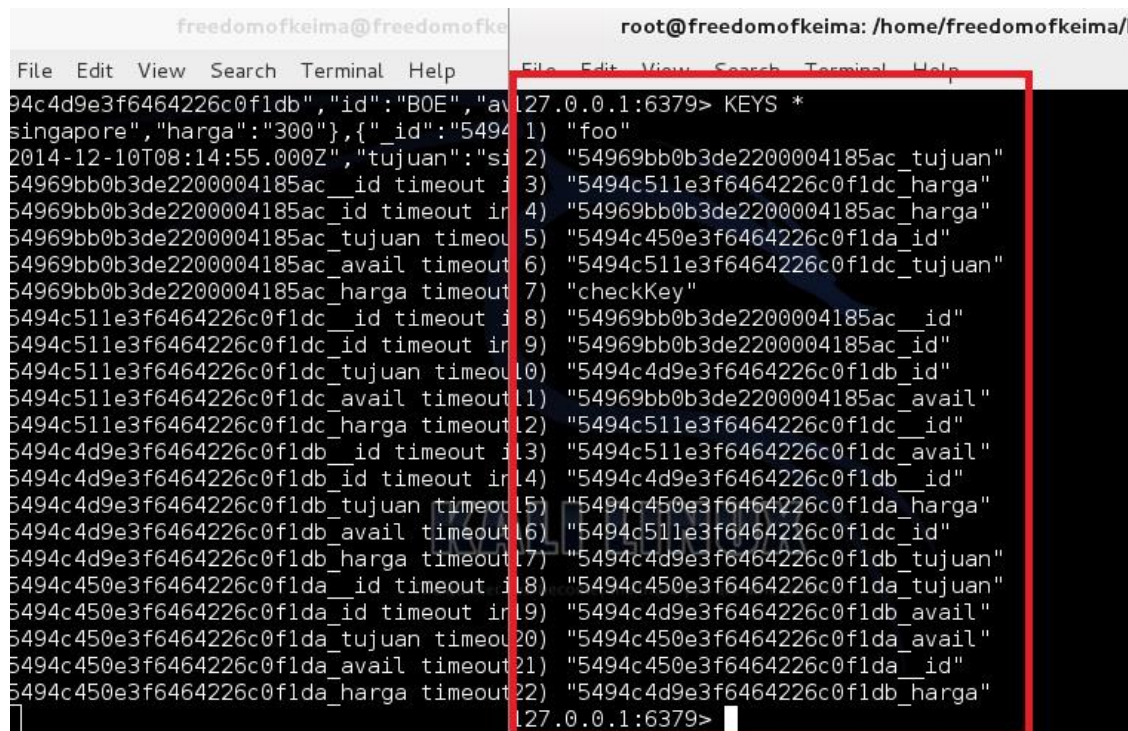
Nomor Kartu Kredit:

0123456789

[Beli](#)

Pada sistem real, sistem ini akan menyimpan informasi pengguna di MongoDB sehingga pengguna tidak perlu lagi memasukkan detail informasi

Kita juga dapat melakukan *cross-checking* terhadap isi *cache layer* menggunakan terminal *redis-client*. Berikut ini adalah isi *keys* dari *cache* yang tersimpan dalam basis data Redis:



```

freedomofkeima@freedomofkeima: /home/freedomofkeima/
File Edit View Search Terminal Help
127.0.0.1:6379> KEYS *
1) "foo"
2) "54969bb0b3de2200004185ac_tujuan"
3) "5494c511e3f6464226c0f1dc_harga"
4) "54969bb0b3de2200004185ac_harga"
5) "5494c450e3f6464226c0f1da_id"
6) "5494c511e3f6464226c0f1dc_tujuan"
7) "checkKey"
8) "54969bb0b3de2200004185ac_id"
9) "54969bb0b3de2200004185ac_id"
10) "5494c4d9e3f6464226c0f1db_id"
11) "54969bb0b3de2200004185ac_avail"
12) "5494c511e3f6464226c0f1dc_id"
13) "5494c511e3f6464226c0f1dc_avail"
14) "5494c4d9e3f6464226c0f1db_id"
15) "5494c450e3f6464226c0f1da_harga"
16) "5494c511e3f6464226c0f1dc_id"
17) "5494c4d9e3f6464226c0f1db_tujuan"
18) "5494c450e3f6464226c0f1da_tujuan"
19) "5494c4d9e3f6464226c0f1db_avail"
20) "5494c450e3f6464226c0f1da_avail"
21) "5494c450e3f6464226c0f1da_id"
22) "5494c4d9e3f6464226c0f1db_harga"
127.0.0.1:6379>

```

Berikut ini adalah kode yang digunakan untuk melakukan *refresh cache* setiap interval waktu tertentu:

```

function refreshCache() {

    var delay = 10000; // API waiting time (10 seconds, assume)

    setTimeout(function() {

        http.get(url, function(http_res) {
            var data = "";
            http_res.on("data", function(chunk) {
                data += chunk;
            });
            http_res.on("end", function() {
                console.log(data);
                data = JSON.parse(data);
                data.forEach(function(entity, i) {
                    // assume all data will be refreshed at same time for this
                    // stage
                    // TODO : Create a list of available keys
                    redis.setex('checkKey', 60, '1', function(error, result) {
                        /** _id */
                        redis.setex(entity.id + ' id', 60, entity.id,

```

```

function(error, result) {
    if (error) console.log('Error: ' + error);
    else console.log(entity._id + ' __id timeout in 60
seconds. ');
    });
    /** id */
    redis.setex(entity._id + '_id', 60, entity.id,
function(error, result) {
    if (error) console.log('Error: ' + error);
    else console.log(entity._id + '_id timeout in 60
seconds. ');
    });
    /** tujuan */
    redis.setex(entity._id + '_tujuan', 60, entity.tujuan,
function(error, result) {
    if (error) console.log('Error: ' + error);
    else console.log(entity._id + '_tujuan timeout in 60
seconds. ');
    });
    /** avail */
    redis.setex(entity._id + '_avail', 60, entity.avail,
function(error, result) {
    if (error) console.log('Error: ' + error);
    else console.log(entity._id + '_avail timeout in 60
seconds. ');
    });
    /** harga */
    redis.setex(entity._id + '_harga', 60, entity.harga,
function(error, result) {
    if (error) console.log('Error: ' + error);
    else console.log(entity._id + '_harga timeout in 60
seconds. ');
    });
    });
    });
    });}, delay);
}

// Timer for refreshing cache
setInterval(function() {refreshCache()}, 10 * 1000); // every 10
seconds, for example, to ensure cache is never empty

```

Sedangkan berikut ini adalah kode yang digunakan untuk mendapatkan informasi tiket, dengan asumsi bahwa pengguna perlu menunggu waktu yang cukup lama apabila data tersebut belum tersimpan dalam *cache*:

```

exports.getTicket = function(req, res) {
    redis.get('checkKey', function(error, result) {
        console.log('checkKey: ' + result);
        if (result != null) { // assume data is available in cache for
60 seconds period in a set (all data)
            loadData()(function (data) {
                res.json({messages: data, fresh: false});
            });
        }
    });
}

```

```
});  
} else { // data is not available in cache  
  setTimeout(function() {  
    loadData()(function (data) {  
      res.json({messages: data, fresh: true});  
    });}, 2500);  
  }  
});  
};
```

#### IV. Konfigurasi aplikasi

Untuk *prototype* aplikasi, kita hanya akan menggunakan aplikasi berikut ini:

1. NodeJS
2. ExpressJS
3. AngularJS
4. Redis
5. MongoDB (Eksternal *web service*)
6. NPM
7. Bower

Amazon Route 53, Amazon Elastic Load Balancer, Amazon EC2, Docker, dan Vagrant dapat diabaikan untuk kasus penanganan permasalahan nomor 2 dan nomor 5.

Konfigurasi aplikasi dapat diakses pada *file* **README** yang terlampir dalam *deliverables*.