

# **Membangun Web API dengan menggunakan JSON sebagai format serialisasi data**

Muhammad Ghazali  
Program Studi Teknik Informatika  
Fakultas Teknik  
Universitas Widyatama  
<muhammad.ghazali@widyatama.ac.id>

27 Mei 2013

# Abstraksi

LayangLayang Mobile (LLM) merupakan salah perusahaan yang bergerak di bidang *mobile application development*. Saat ini LayangLayang Mobile sedang mengembangkan sebuah produk bernama CampusLife. Produk yang dikembangkan tersebut merupakan aplikasi *mobile* yang bertujuan untuk membantu civitas kampus mengakses informasi relevan tentang kampus mereka.

Setiap informasi yang ditampilkan melalui aplikasi *mobile* CampusLife merupakan data yang sudah diolah dan diambil dari Web API<sup>1</sup> CampusLife. Saat ini LLM belum memiliki Web API tersebut. Berdasarkan kondisi tersebut, penulis bekerjasama dengan LLM untuk membangun Web API CampusLife. Web API yang akan dibangun bertujuan untuk membuka akses secara tidak langsung ke *data store*<sup>2</sup> yang tersimpan di salah satu layanan *Database as a Service*<sup>3</sup> yang digunakan oleh LLM di AppFog<sup>4</sup>. Seluruh data-data *event* yang tersimpan di *data store* akan diolah oleh Web API menjadi data dengan format yang dapat dikonsumsi dengan mudah oleh aplikasi *mobile* CampusLife. Proses pengolahan tersebut dinamakan serialisasi data<sup>5</sup>.

Dalam penelitian ini penulis akan memilih format serialisasi data JSON untuk digunakan merepresentasikan setiap data-data *event* dalam format yang dapat dikonsumsi oleh aplikasi *mobile* CampusLife. Penulis memilih format serialisasi data JSON karena JSON lebih mudah dibaca ditulis dan dibaca oleh mesin (komputer) dan manusia. Selain itu JSON lebih mudah untuk diproses karena memiliki struktur yang lebih sederhana dibandingkan XML[6][3].

Kata kunci: Web API, JSON, Format Serialisasi Data

---

<sup>1</sup>[http://en.wikipedia.org/wiki/Web\\_API](http://en.wikipedia.org/wiki/Web_API)

<sup>2</sup>[http://en.wikipedia.org/wiki/Data\\_store](http://en.wikipedia.org/wiki/Data_store)

<sup>3</sup>[http://en.wikipedia.org/wiki/Cloud\\_database](http://en.wikipedia.org/wiki/Cloud_database)

<sup>4</sup><http://www.appfog.com/>

<sup>5</sup>Lihat bagian Landasan teori: Serialiasi Data

# **Kata Pengantar**

# Daftar Isi

<b>Abstraksi</b>	<b>1</b>
<b>Kata Pengantar</b>	<b>2</b>
<b>Daftar Gambar</b>	<b>6</b>
<b>Daftar Tabel</b>	<b>7</b>
<b>1 Pendahuluan</b>	<b>9</b>
1.1 Latar Belakang dan Masalah . . . . .	9
1.2 Rumusan Masalah . . . . .	10
1.3 Tujuan dan Manfaat Penelitian . . . . .	10
1.4 Batasan Masalah . . . . .	11
1.5 Metodologi Penelitian . . . . .	11
1.6 Sistematika Penulisan . . . . .	12
<b>2 Landasan Teori</b>	<b>13</b>
2.1 Web API . . . . .	13
2.1.1 Apa itu Web API . . . . .	13
2.1.2 Kenapa Web API . . . . .	13
2.1.3 RESTful Web API . . . . .	14
2.2 Serialisasi data . . . . .	15
2.3 JSON . . . . .	16
2.3.1 Pengantar . . . . .	16
2.3.2 Struktur . . . . .	17
2.3.3 Tipe Data yang Didukung JSON . . . . .	17
2.3.4 Contoh-contoh . . . . .	20
2.4 XML . . . . .	21
2.4.1 Pengantar . . . . .	21
2.4.2 Contoh Dokumen XML . . . . .	21

2.4.3	Kritik Terhadap XML . . . . .	22
2.5	Menggunakan JSON daripada XML . . . . .	22
2.6	Agile Software Development . . . . .	23
2.6.1	Pengantar . . . . .	23
2.6.2	Manifesto Pengembangan Perangkat Lunak Agile . . . . .	23
2.6.3	Prinsip-prinsip di balik Manifesto Agile . . . . .	24
2.6.4	Bagaimana Agile Berbeda . . . . .	25
2.6.5	Beberapa Praktek Agile yang Diterapkan . . . . .	26
2.6.6	Mempraktekkan Test-Driven Development . . . . .	26
2.6.7	Hasil Akhir Implementasi . . . . .	28
<b>3</b>	<b>Analisis Sistem</b>	<b>29</b>
3.1	Roadmap Pengerjaan Purwa-rupa Web API . . . . .	29
3.2	High-level Architecture CampusLife . . . . .	30
3.2.1	Presentation Tier . . . . .	30
3.2.2	Logic Tier . . . . .	31
3.2.3	Data Tier . . . . .	31
3.2.4	Keuntungan dari Arsitektur Three-tier . . . . .	31
3.2.5	Ruang Lingkup Pengerjaan . . . . .	31
3.2.6	Daftar Kebutuhan yang harus dipenuhi . . . . .	32
3.2.7	Pola Alur Kerja Ketika Mengumpulkan Daftar Kebutuhan . . . . .	32
3.2.8	Acceptance Criteria . . . . .	34
<b>4</b>	<b>Perancangan Sistem</b>	<b>35</b>
4.1	Arsitektur Teknis Web API . . . . .	35
4.2	Service Interaction Model . . . . .	37
4.2.1	Permintaan Detail Event dalam format JSON . . . . .	38
4.2.2	Permintaan Detail Event dalam format XML . . . . .	39
4.2.3	Permintaan Daftar Event dalam format JSON . . . . .	40
4.2.4	Permintaan Daftar Event dalam format XML . . . . .	41
4.3	Struktur URI . . . . .	42
4.4	Rancangan Skema Representasi Resource . . . . .	42
4.4.1	Pertimbangan Rancangan Skema Representasi Resource dalam format JSON . . . . .	42
<b>5</b>	<b>Implementasi dan Pengujian Sistem</b>	<b>44</b>
5.1	Implementasi Web API . . . . .	44
5.1.1	Teknologi yang Dipakai untuk Membangun Web API . . . . .	44

5.2	Pengujian Web API . . . . .	45
5.2.1	Skenario Pengujian . . . . .	45
5.3	Hasil Pembandingan Ukuran Data JSON dan XML . . . . .	45
<b>6</b>	<b>Penutup</b>	<b>47</b>
6.1	Kesimpulan . . . . .	47
6.2	Saran . . . . .	47
<b>A</b>	<b>Acceptance Criteria</b>	<b>50</b>
A.1	Get Event Details Acceptance Criteria . . . . .	50
A.1.1	Scenario 1: Event Details request should return response in JSON . . . . .	50
A.1.2	Scenario 2: Event Details request should return response in XML . . . . .	50
A.1.3	Scenario 3: Unsupported Content-Type should return HTTP 406 code . . . . .	51
A.2	Get Event List Acceptance Criteria . . . . .	51
A.2.1	Scenario 1: Event List request should return response in JSON	51
A.2.2	Scenario 2: Event List request should return response in XML	51
A.2.3	Scenario 3: Unsupported Content-Type should return HTTP 406 code . . . . .	51
<b>B</b>	<b>Detail Rancangan Skema Representasi Resource</b>	<b>53</b>
B.1	Rancangan Skema Representasi Resource Detail Event . . . . .	53
B.1.1	Rancangan Skema Representasi Resource Detail Event dalam format JSON . . . . .	53
B.1.2	Rancangan Skema Representasi Resource Detail Event dalam format XML . . . . .	54
B.2	Rancangan Skema Representasi Resource Daftar Event . . . . .	55
B.2.1	Rancangan Skema Representasi Resource Daftar Event dalam format XML . . . . .	56
<b>C</b>	<b>Detail Respon Web API</b>	<b>59</b>
C.1	Hasil Respon Permintaan Detail Event . . . . .	59
C.1.1	Hasil Respon Permintaan Detail Event dalam Format JSON	59
C.1.2	Hasil Respon Permintaan Detail Event dalam Format XML .	60
C.2	Hasil Respon Permintaan Mengambil Daftar Event . . . . .	62
C.2.1	Hasil Respon Permintaan Daftar Event dalam Format JSON	62
C.2.2	Hasil Respon Permintaan Daftar Event dalam Format XML	65

<b>D Hasil Pengujian Web API</b>	<b>68</b>
<b>E Hasil Pembandingan Ukuran Data</b>	<b>71</b>
E.1 Empty Resource . . . . .	71
E.2 Normal Resource . . . . .	72
E.3 Just Once . . . . .	72

# Daftar Gambar

2.1	RESTful Web API . . . . .	14
2.2	Proses Serialisasi . . . . .	15
2.3	Diagram Sintaks untuk Tipe Object [6] . . . . .	17
2.4	Diagram Sintaks untuk Tipe Array [6] . . . . .	18
2.5	Diagram Sintaks untuk Tipe String [6] . . . . .	18
2.6	Diagram Sintaks untuk Tipe Number [6] . . . . .	19
2.7	Ilustrasi aktivitas kontinu di <i>Agile</i> [21] . . . . .	26
2.8	Bagaimana TDD Bekerja [21] . . . . .	27
3.1	High-level architecture CampusLife . . . . .	30
4.1	Model Arsitektur Teknis Web API dalam Free-form Diagram . . . . .	36
4.2	Permintaan Detail Event dalam Format JSON . . . . .	38
4.3	Permintaan Detail Event dalam Format XML . . . . .	39
4.4	Permintaan Daftar Event dalam Format JSON . . . . .	40
4.5	Permintaan Daftar Event dalam Format XML . . . . .	41
4.6	Struktur URI Web API . . . . .	42
E.1	Perbandingan Jumlah Permintaan yang berhasil direspon untuk skenario Empty Resource . . . . .	71
E.2	Perbandingan Jumlah Permintaan yang berhasil direspon untuk skenario Normal Resource . . . . .	72



## **Daftar Tabel**

# Bab 1

## Pendahuluan

### 1.1 Latar Belakang dan Masalah

CampusLife adalah *mobile information directory application* yang dikembangkan oleh LayangLayang Mobile (LLM) untuk menyediakan informasi yang relevan kepada civitas kampus. Salah satu fitur utama yang akan dirilis dalam waktu dekat adalah menyediakan informasi *event-event* terbaru kepada civitas kampus.

Setiap informasi yang ditampilkan melalui aplikasi *mobile* CampusLife merupakan data yang sudah diolah dan diambil dari Web API<sup>1</sup> CampusLife. Saat ini LLM belum memiliki Web API dan penulis berniat untuk membangun Web API tersebut. Web API yang akan dibangun bertujuan untuk membuka akses secara tidak langsung ke *data store*<sup>2</sup> yang tersimpan di salah satu layanan *Database as a Service*<sup>3</sup> yang digunakan oleh LLM di AppFog<sup>4</sup>. Seluruh data-data *event* yang tersimpan di *data store* akan diolah oleh Web API menjadi data dengan format yang dapat dikonsumsi dengan mudah oleh aplikasi *mobile* CampusLife. Proses pengolahan tersebut dinamakan serialisasi data<sup>5</sup>.

Dalam penelitian ini penulis akan memilih format serialisasi data JSON untuk digunakan merepresentasikan setiap data-data *event* dalam format yang dapat dikonsumsi oleh aplikasi *mobile* CampusLife. Penulis memilih format serialisasi data JSON karena JSON lebih mudah dibaca ditulis dan dibaca oleh mesin (komputer) dan manusia. Selain itu JSON lebih mudah untuk diproses karena memiliki struktur yang lebih sederhana dibandingkan XML[6][3].

*Smartphone* sebagai perangkat tempat aplikasi *mobile* CampusLife berjalan,

---

<sup>1</sup>[http://en.wikipedia.org/wiki/Web\\_API](http://en.wikipedia.org/wiki/Web_API)

<sup>2</sup>[http://en.wikipedia.org/wiki/Data\\_store](http://en.wikipedia.org/wiki/Data_store)

<sup>3</sup>[http://en.wikipedia.org/wiki/Cloud\\_database](http://en.wikipedia.org/wiki/Cloud_database)

<sup>4</sup><http://www.appfog.com/>

<sup>5</sup>Lihat bagian Landasan teori: Serialiasi Data

merupakan salah satu *mobile computing devices* yang memiliki masa hidup baterai dan ketersediaan *bandwidth* yang terbatas.[1]. Dengan kedua keterbatasan tersebut, dalam penelitian ini penulis akan mengkaji penerapan format serialisasi data JSON yang efektif untuk menghasilkan ukuran data yang optimal saat pengiriman data berlangsung dari Web API ke aplikasi *mobile* CampusLife. Hasil akhir yang diharapkan adalah format serialisasi data JSON mampu menghasilkan ukuran data yang optimal untuk dikonsumsi oleh aplikasi *mobile*.

## 1.2 Rumusan Masalah

Berdasarkan latar belakang masalah yang telah dipaparkan, maka masalah yang akan diteliti dalam penelitian ini dapat dirumuskan sebagai berikut:

1. Bagaimana membangun Web API dengan efektif agar data yang dikirimkan dapat memiliki ukuran data yang optimal untuk dikonsumsi oleh aplikasi *mobile* CampusLife?
2. Bagaimana JSON dapat diterapkan secara efektif agar data yang dihasilkan dapat memiliki ukuran data yang optimal untuk dikonsumsi oleh aplikasi *mobile* CampusLife?

## 1.3 Tujuan dan Manfaat Penelitian

Tujuan dari pelaksanaan penelitian tugas akhir yang dilakukan penulis adalah:

1. Membangun purwa-rupa Web API yang mampu mengirimkan data dengan ukuran data yang optimal untuk dikonsumsi oleh aplikasi *mobile* CampusLife.
2. Menerapkan JSON dengan efektif untuk menghasilkan data dengan ukuran data yang optimal untuk dikonsumsi oleh aplikasi *mobile* CampusLife.

Penulis mengharapkan hasil penelitian ini akan membawa manfaat positif bagi kepentingan dunia akademik dan praktis dalam hal penerapan format serialisasi data JSON untuk menghasilkan ukuran data yang optimal untuk dikonsumsi oleh aplikasi *mobile*.

## 1.4 Batasan Masalah

Untuk memperjelas ruang lingkup pelaksanaan penelitian, penulis memiliki batasan masalah meliputi:

1. Pembangunan Web API hanya akan sampai pada tahap purwa-rupa.
2. Pembangunan Web API hanya akan meliputi API untuk mengambil data-data *event*.
3. JSON hanya mampu merepresentasikan data dalam bentuk teks, oleh karena itu data yang akan digunakan hanya terbatas pada data yang berbasis teks.
4. Skema data *event* akan disediakan oleh pihak LLM.
5. Dikarenakan keterbatasan waktu yang dimiliki oleh penulis, maka demo untuk mengakses Web API tidak jadi dilakukan melalui aplikasi *mobile*.
6. Penulis akan melakukan demo untuk mengakses Web API melalui Apache Benchmark<sup>6</sup> dalam lingkungan lokal yang terisolasi.
7. Tidak membahas mengenai keamanan perangkat lunak, data dan jaringan.
8. Pengembangan perangkat lunak menggunakan sebagian praktek dari *Agile* dan tidak akan membahas *Agile* secara komprehensif.

## 1.5 Metodologi Penelitian

Adapun tahapan penelitian yang akan dilakukan oleh penulis tahap-tahap berikut:

1. Identifikasi masalah. Pada tahap ini penulis merumuskan masalah yang akan diteliti.
2. Perumusan hipotesis. Pada tahap ini penulis merumuskan jawaban sementara dari permasalahan yang diteliti.
3. Pengujian hipotesis. Pada tahap ini penulis menguji penerapan JSON terhadap Web API yang sudah dibangun dan pembangunan perangkat lunak akan dilakukan dengan menggunakan metodologi pengembangan perangkat lunak *Agile*.
4. Kesimpulan. Pada tahap ini penulis menganalisa hasil pengujian hipotesis.

---

<sup>6</sup><http://httpd.apache.org/docs/2.4/programs/ab.html>

Adapun cara untuk menunjang penelitian dilakukan dengan melakukan studi kepustakaan, yaitu pengumpulan data dari artikel-artikel, jurnal dan buku-buku yang berhubungan dengan topik pembahasan di penelitian.

## **1.6 Sistematika Penulisan**

Sistematika pembahasan laporan ini terdiri dari enam bab, yaitu:

**Bab I Pendahuluan** Pada bagian ini berisikan pendahuluan laporan yang berisi latar belakang, rumusan masalah, tujuan dan manfaat penelitian, batasan masalah, metode penelitian dan sistematika penulisan laporan.

**Bab II Landasan Teori** Pada bagian ini akan dibahas landasan teori yang berkaitan dan digunakan selama masa penelitian.

**Bab III Analisis Sistem** Pada bagian ini akan dibahas hasil analisis terhadap masalah.

**Bab IV Perancangan Sistem** Pada bagian ini akan dibahas hasil perancangan sistem.

**Bab V Implementasi dan Pengujian Sistem** Pada bagian ini akan dibahas proses implementasi dan pengujian sistem berdasarkan kriteria pengujian yang sudah ditentukan.

**Bab VI Penutup** Pada bagian ini berisikan kesimpulan dan saran yang didapatkan dari hasil penelitian.

# Bab 2

## Landasan Teori

### 2.1 Web API

#### 2.1.1 Apa itu Web API

*Application programming interface* (API) adalah sebuah protokol yang dimaksudkan untuk digunakan sebagai antarmuka oleh komponen perangkat lunak untuk berkomunikasi satu sama lain. Ketika digunakan dalam konteks pengembangan *web*, API biasanya didefinisikan sebagai satu set *Hypertext Transfer Protocol* (HTTP) *request message*, dan disertai dengan *response message* dalam *Extensible Markup Language* atau *JavaScript Object Notation*.

Praktek penerbitan Web API telah memungkinkan masyarakat *web* seperti pengembang aplikasi berbasis *web* untuk membuat arsitektur terbuka untuk berbagi konten dan data antara masyarakat dan aplikasi. Dengan cara ini, konten yang dibuat dalam satu tempat dapat dikirimkan secara dinamis dan diperbaharui di beberapa lokasi di *web* [9].

#### 2.1.2 Kenapa Web API

Ketika menjalankan aplikasi *web*, pada saat tertentu kita ingin menyediakan Web API. Berikut adalah beberapa alasan yang umum kenapa perlu menyediakan Web API [11]:

1. Untuk memungkinkan pengguna mengakses data mereka melalui aplikasi pihak ketiga. Pertimbangkan berapa banyak pihak ketiga Twitter klien yang ada<sup>1</sup>, semua menggunakan API Twitter<sup>2</sup>, daripada mengakses langsung ke si-

---

<sup>1</sup>[http://en.wikipedia.org/wiki/List\\_of\\_Twitter\\_services\\_and\\_applications](http://en.wikipedia.org/wiki/List_of_Twitter_services_and_applications)

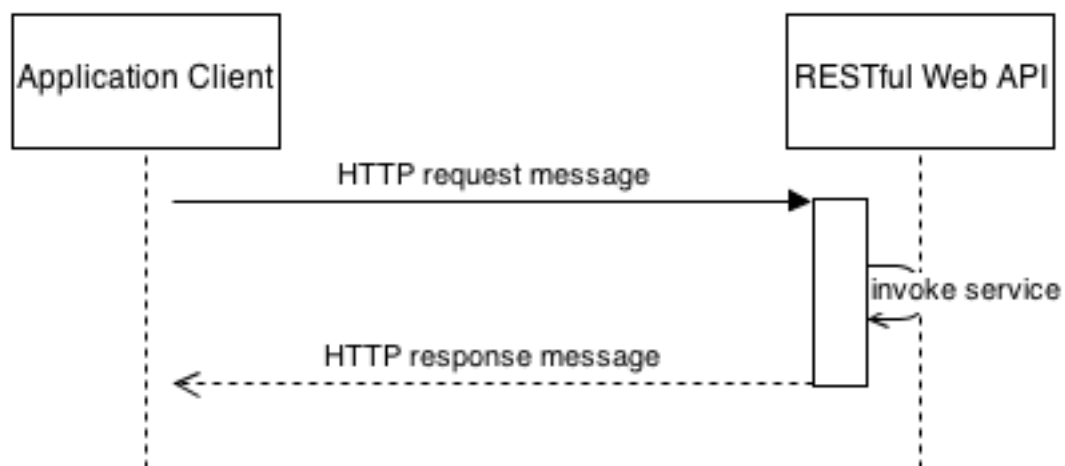
<sup>2</sup><https://dev.twitter.com/>

tus *web*<sup>3</sup>. Hal yang sama berlaku untuk Amazon<sup>4</sup> dan eBay<sup>5</sup>, antara lain, yang memungkinkan pengguna untuk mengakses data katalog mereka dan bahkan mengeksekusi penjualan, semua melalui API.

2. Untuk memungkinkan pengembang aplikasi mobile untuk mengakses situs dengan mengirim dan mengambil data menggunakan HTTP.
3. Untuk memungkinkan aplikasi untuk mengakses data sendiri melalui panggilan Ajax<sup>6</sup>. Bila membuat panggilan JavaScript di latar belakang menggunakan Ajax, kemungkinan besar aplikasi akan menggunakan panggilan API, untuk menerima XML atau JSON, bukan HTML yang membutuhkan *parsing* lebih lanjut.

### 2.1.3 RESTful Web API

RESTful Web API adalah Web API yang diimplementasikan dengan menggunakan HTTP dan prinsip-prinsip REST<sup>7</sup>. *Representational State Transfer* (REST) mendefinisikan seperangkat prinsip arsitektur dimana penulis dapat merancang layanan Web yang berfokus pada sistem *resource*, termasuk bagaimana keadaan *resource* dipanggil dan ditransfer melalui HTTP oleh berbagai macam klien yang ditulis dalam bahasa (pemrograman) yang berbeda. Berikut adalah diagram yang mengilustrasikan proses saat klien mengirimkan *request* dan RESTful Web API mengirimkan *response* kepada klien:



Gambar 2.1: RESTful Web API

<sup>3</sup><https://twitter.com/>

<sup>4</sup><http://www.amazon.com/>

<sup>5</sup><http://www.ebay.com/>

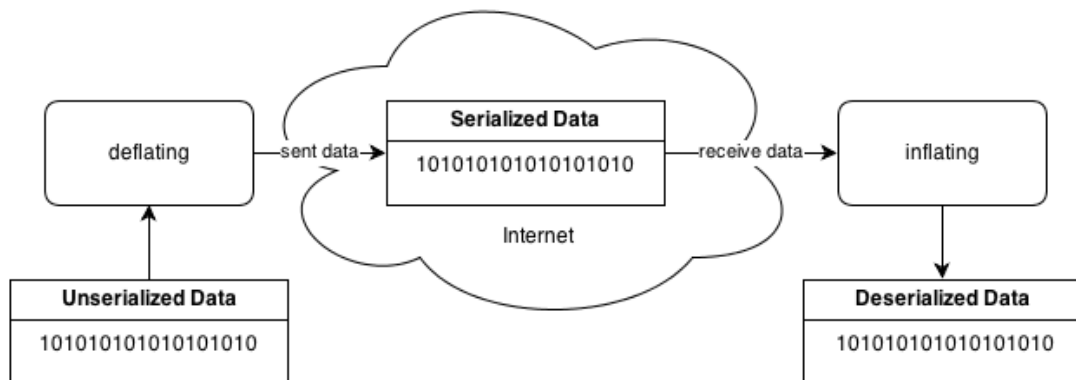
<sup>6</sup><https://developer.mozilla.org/en/docs/AJAX>

<sup>7</sup><http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>

Jika diukur dengan jumlah layanan Web yang menggunakannya, REST telah muncul dalam beberapa tahun terakhir sebagai model desain layanan Web yang dominan. Bahkan, REST memiliki dampak besar di dunia Web yang telah sebagian besar menggantikan desain antarmuka berbasis SOAP-WSDL dan karena REST memiliki gaya yang jauh lebih sederhana untuk digunakan [12]. Berdasarkan informasi yang penulis dapatkan, pada saat laporan ini ditulis sudah lebih dari 5000 REST Web API yang diimplementasikan di dunia nyata<sup>8</sup>.

## 2.2 Serialisasi data

Dalam ilmu komputer, dalam konteks penyimpanan data dan transmisi, serialisasi (alias *deflating*) adalah proses menerjemahkan struktur basis data atau kondisi objek ke dalam format yang dapat disimpan lebih lanjut dalam basis data (misalnya, dalam sebuah berkas atau *buffer* memori, atau ditransmisikan melalui koneksi jaringan) dan dilakukan proses deserialisasi (alias *inflating*) kembali di komputer lain [4]. Proses serialisasi dan deserialisasi diilustrasikan dalam Gambar 2.2.



Gambar 2.2: Proses Serialisasi

Pada saat penulis membuat laporan, sudah ada lebih dari 10 format serialisasi data<sup>9</sup>:

1. ASN.1
2. Bencode
3. *Candle Markup*
4. *Comma-separated values (CSV)*

<sup>8</sup>Web Services Directory <http://www.programmableweb.com/apis/directory/1?protocol=REST>

<sup>9</sup>[http://en.wikipedia.org/wiki/Comparison\\_of\\_data\\_serialization\\_formats](http://en.wikipedia.org/wiki/Comparison_of_data_serialization_formats)



5. BSON
6. *D-Bus Message Protocol*
7. JSON
8. MessagePack
9. Netstrings
10. OGD
11. *Property list*
12. *Protocol Buffers*
13. S-expressions
14. Sereal
15. *Structured Data eXchange Formats*
16. Thrift
17. *eXternal Data Representation*
18. XML
19. XML-RPC
20. YAML

## 2.3 JSON

### 2.3.1 Pengantar

*JavaScript Object Notation* (JSON) adalah format *data interchange* ringan berbasis teks yang bersifat *language-independent*<sup>10</sup>. Relatif sangat mudah bagi manusia untuk membaca dan menulis JSON. Relatif sangat mudah pula bagi mesin (komputer) untuk mengurai dan menghasilkan JSON.

---

<sup>10</sup>Dengan sifat seperti ini JSON bisa digunakan di banyak bahasa pemrograman, selama ada dukungan *library* untuk bahasa pemrograman yang akan digunakan

JSON dibuat dengan berbasiskan JavaScript, standar ECMA-262 Edisi 3 - Desember 1999<sup>11</sup>. JSON merupakan format teks yang benar - benar bahasa independen tetapi menggunakan konvensi yang akrab bagi *programmer* dari keluarga bahasa C, termasuk C, C + +, C#, Java, JavaScript, Perl, Python, dan banyak lainnya. Properti ini membuat JSON sebagai bahasa *data interchange* yang ideal[13].

### 2.3.2 Struktur

JSON dibangun di atas dua struktur:

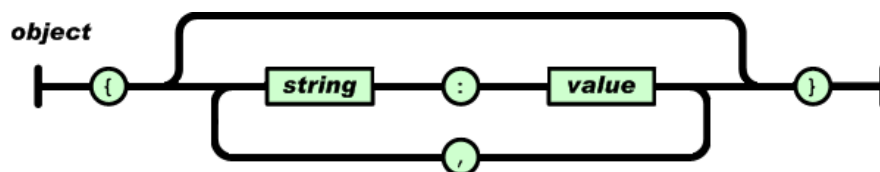
1. *Collection of name/value pairs*. Dalam berbagai bahasa, struktur ini direalisasikan sebagai *object*, *record*, *struct*, *dictionary*, *hash table*, *keyed list*, atau *associative array*.
2. *Ordered List*. Dalam kebanyakan bahasa, struktur ini direalisasikan sebagai *array*, *vector*, *list*, atau *sequence*.

Kedua struktur tersebut adalah struktur data universal. Hampir semua bahasa pemrograman modern mendukung struktur tersebut dalam satu bentuk atau lain [13].

### 2.3.3 Tipe Data yang Didukung JSON

JSON dapat mewakili empat tipe primitif (*string*, *number*, *boolean*, and *null*) dan dua tipe terstruktur (*object* and *array*) [14].

Sebuah *object* adalah sebuah *unordered collection* yang terdiri dari nol atau lebih pasangan nama/nilai, dimana sebuah nama adalah *string* dan sebuah nilai adalah *string*, *number*, *boolean*, *null*, *object*, atau *array*.



Gambar 2.3: Diagram Sintaks untuk Tipe Object [6]

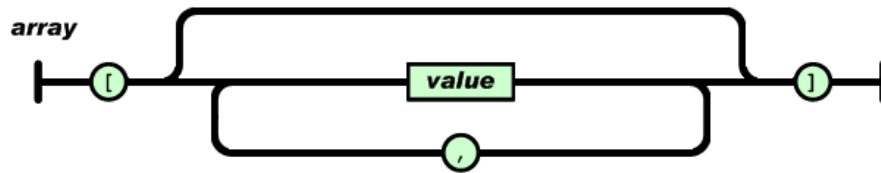
*Array* adalah rangkaian terurut yang terdiri dari nol atau lebih nilai.

*Value* dapat berupa *string* dalam tanda kutip ganda, *number*, *boolean*, *object* atau *array*. Struktur ini dapat diulang.

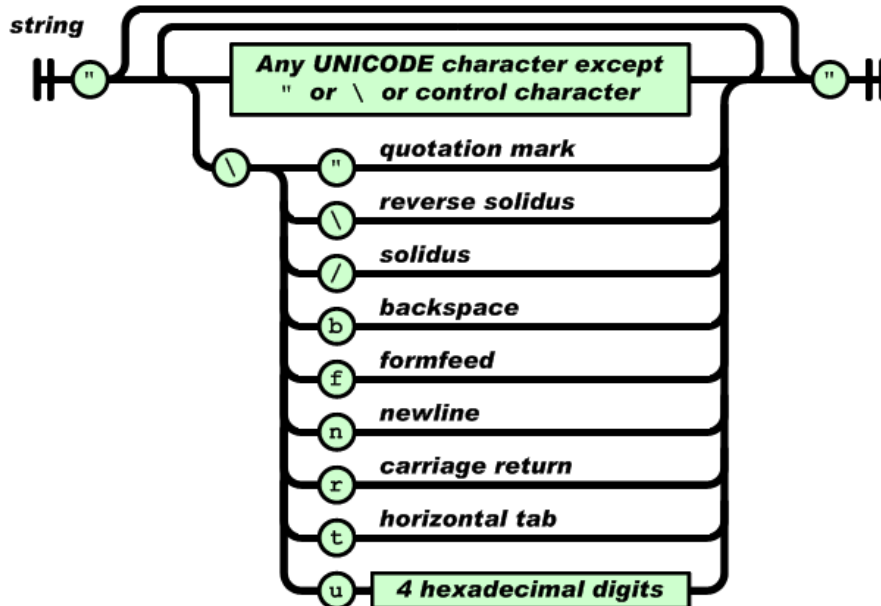
*String* adalah rangkaian yang terdiri dari nol atau lebih karakter *Unicode*<sup>12</sup>.

<sup>11</sup><http://www.ecma-international.org/publications/standards/Ecma-262-arch.htm>

<sup>12</sup><http://www.unicode.org/versions/Unicode6.2.0/>

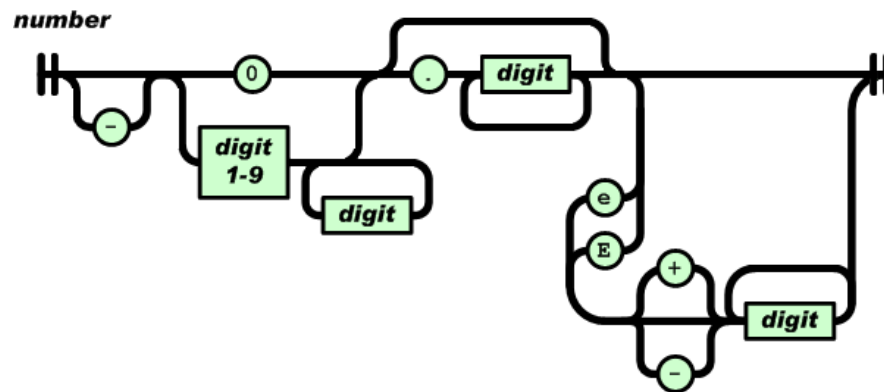


Gambar 2.4: Diagram Sintaks untuk Tipe Array [6]



Gambar 2.5: Diagram Sintaks untuk Tipe String [6]

*Number* di JSON sama seperti tipe data *number* di C atau Java, kecuali format oktal<sup>13</sup> dan heksadesimal<sup>14</sup> tidak digunakan.



Gambar 2.6: Diagram Sintaks untuk Tipe Number [6]

<sup>13</sup>Contoh angka dalam bentuk oktal 061 062 063

<sup>14</sup>Contoh angka dalam bentuk 9B0

### 2.3.4 Contoh-contoh

Ini adalah contoh *object* dalam format JSON:

```
{
  "Image": {
    "Width": 800,
    "Height": 600,
    "Title": "View from 15th Floor",
    "Thumbnail": {
      "Url": "http://www.example.com/image/481989943",
      "Height": 125,
      "Width": "100"
    },
    "IDs": [116, 943, 234, 38793]
  }
}
```

Ini adalah contoh sebuah *array* dalam format JSON yang berisi 2 *object*:

```
[
  {
    "precision": "zip",
    "Latitude": 37.7668,
    "Longitude": -122.3959,
    "Address": "",
    "City": "SAN FRANCISCO",
    "State": "CA",
    "Zip": "94107",
    "Country": "US"
  },
  {
    "precision": "zip",
    "Latitude": 37.371991,
    "Longitude": -122.026020,
    "Address": "",
    "City": "SUNNYVALE",
    "State": "CA",
    "Zip": "94085",
    "Country": "US"
  }
]
```

```
}  
]
```

## 2.4 XML

### 2.4.1 Pengantar

*Extensible Markup Language* (XML) adalah bahasa *markup* yang mendefinisikan seperangkat aturan untuk *document encoding* dalam format yang dapat terbaca oleh manusia dan mesin. Ini didefinisikan dalam spesifikasi XML 1.0<sup>15</sup> yang dihasilkan oleh W3C.

Tujuan desain XML menekankan kesederhanaan, keumuman, dan kegunaan melalui Internet. XML adalah format data tekstual dengan dukungan kuat melalui Unicode. Meskipun desain XML berfokus pada dokumen, namun secara luas digunakan untuk menrepresentasikan struktur data yang berubah-ubah, misalnya dalam layanan web [16].

Dikarenakan topik penelitian ini tidak membandingkan antara JSON versus XML dan tidak fokus ke XML, maka penulis hanya menjelaskan XML secara singkat.

### 2.4.2 Contoh Dokumen XML

Berikut adalah contoh dokumen XML dari tutorial W3Schools<sup>16</sup>:

```
<?xml version="1.0" encoding="ISO-8859-1"?>  
<CATALOG>  
  <CD>  
    <TITLE>Empire Burlesque </TITLE>  
    <ARTIST>Bob Dylan </ARTIST>  
    <COUNTRY>USA</COUNTRY>  
    <COMPANY>Columbia </COMPANY>  
    <PRICE>10.90 </PRICE>  
    <YEAR>1985 </YEAR>  
  </CD>  
  <CD>  
    <TITLE>Hide your heart </TITLE>  
    <ARTIST>Bonnie Tyler </ARTIST>
```

<sup>15</sup><http://www.w3.org/TR/REC-xml/>

<sup>16</sup>[http://www.w3schools.com/xml/cd\\_catalog.xml](http://www.w3schools.com/xml/cd_catalog.xml)

```
<COUNTRY>UK</COUNTRY>
<COMPANY>CBS Records </COMPANY>
<PRICE>9.90</PRICE>
<YEAR>1988</YEAR>

</CD>
</CATALOG>
```

### 2.4.3 Kritik Terhadap XML

XML telah sering dikritik karena bertele-tele dan kerumitan yang dimiliki. Pemetaan model pohon dasar XML untuk tipe sistem bahasa pemrograman atau basis data bisa menjadi sulit, terutama ketika XML digunakan untuk pertukaran data yang sangat terstruktur antara aplikasi, yang dimana bukan tujuan desain utama XML [16].

Walaupun XML bisa digunakan untuk *data interchange*, namun, XML tidak cocok untuk *data interchange*. Ini membawa banyak bagasi (baca: pengeluaran tambahan), dan tidak cocok dengan model data dari kebanyakan bahasa pemrograman. Ketika kebanyakan *programmer* melihat XML untuk pertama kalinya, mereka terkejut melihat betapa jelek dan tidak efisiennya XML [6].

## 2.5 Menggunakan JSON daripada XML

Pada dasarnya setiap hal selalu memiliki kelebihanannya dan kekurangannya serta *use case* masing-masing. Seperti yang sudah dijelaskan sebelumnya tentang kritik terhadap XML, pada bagian ini dijelaskan beberapa poin kenapa lebih memilih untuk menggunakan JSON daripada XML. Berikut adalah beberapa alasan tersebut:

1. JSON lebih sederhana
  - (a) JSON lebih sederhana dibandingkan XML. JSON memiliki tata bahasa yang jauh lebih kecil dan lebih mudah dipetakan langsung ke struktur data yang digunakan dalam bahasa pemrograman modern.
  - (b) JSON jauh lebih mudah bagi manusia untuk dibaca daripada XML. JSON juga lebih mudah untuk ditulis dan dibaca oleh mesin.
2. JSON dapat dipetakan dengan lebih mudah untuk sistem berorientasi objek.
3. JSON sudah diadopsi secara luas oleh banyak pihak di industri perangkat lunak [17][18].

## 2.6 Agile Software Development

### 2.6.1 Pengantar

Pengembangan perangkat lunak *Agile* adalah sekelompok metode pengembangan perangkat lunak berdasarkan metode pengembangan *iterative and incremental*<sup>17</sup>, di mana persyaratan dan solusi berkembang melalui kolaborasi. Ini mempromosikan perencanaan adaptif, perkembangan dan pengiriman yang evolusioner, pendekatan *time-boxed*<sup>18</sup> berulang, dan mendorong respon cepat dan fleksibel terhadap perubahan [5].

### 2.6.2 Manifesto Pengembangan Perangkat Lunak Agile

Pada Februari 2001, 17 pengembang perangkat lunak<sup>19</sup> bertemu di Snowbird, Utah, untuk membahas metode pengembangan ringan. Mereka menerbitkan manifesto untuk *Agile Software Development* untuk menentukan pendekatan yang sekarang dikenal sebagai pengembangan perangkat lunak *Agile* [5].

Melalui manifesto tersebut mereka menghargai [20]:

1. Individu dan interaksi lebih dari proses dan sarana perangkat lunak
2. Perangkat lunak yang bekerja lebih dari dokumentasi yang menyeluruh
3. Kolaborasi dengan klien lebih dari negosiasi kontrak
4. Tanggap terhadap perubahan lebih dari mengikuti rencana

Demikian, walaupun mereka menghargai hal di sisi kanan, mereka lebih menghargai hal di sisi kiri. Arti dari item manifesto di sebelah kiri dalam konteks pembangunan perangkat lunak *Agile* dijelaskan di bawah ini:

1. Individu dan interaksi. Dalam pengembangan perangkat lunak *Agile*, mengorganisir diri dan motivasi merupakan hal yang penting, seperti interaksi seperti *co-location*<sup>20</sup> dan *pair programming*<sup>21</sup>.

---

<sup>17</sup>[http://en.wikipedia.org/wiki/Iterative\\_and\\_incremental\\_development](http://en.wikipedia.org/wiki/Iterative_and_incremental_development)

<sup>18</sup><http://en.wikipedia.org/wiki/Timeboxing>

<sup>19</sup>Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Stephen J. Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas

<sup>20</sup>[http://en.wikipedia.org/wiki/Colocation\\_\(business\)](http://en.wikipedia.org/wiki/Colocation_(business))

<sup>21</sup>[http://en.wikipedia.org/wiki/Pair\\_programming](http://en.wikipedia.org/wiki/Pair_programming)



2. Perangkat lunak yang bekerja (berfungsi dengan baik). Perangkat lunak yang bekerja akan lebih bermanfaat dan diterima daripada sekedar menyajikan dokumen untuk klien dalam pertemuan.
3. Kolaborasi dengan klien. Daftar kebutuhan perangkat lunak tidak dapat dikumpulkan secara menyeluruh pada awal siklus pengembangan perangkat lunak, sehingga keterlibatan terus menerus pelanggan atau pemangku kepentingan sangat penting.
4. Tanggap terhadap perubahan. Pengembangan perangkat lunak *Agile* difokuskan pada respon cepat terhadap perubahan dan pembangunan berkelanjutan.

### 2.6.3 Prinsip-prinsip di balik Manifesto Agile

Manifesto pengembangan perangkat lunak *Agile* didasarkan pada dua belas prinsip:

1. Prioritas utama kami adalah memuaskan klien dengan menghasilkan perangkat lunak yang bernilai secara dini dan rutin.
2. Menyambut perubahan kebutuhan, walaupun terlambat dalam pengembangan perangkat lunak. Proses *Agile* memanfaatkan perubahan untuk keuntungan kompetitif klien.
3. Menghasilkan perangkat lunak yang bekerja secara rutin, dari jangka waktu beberapa minggu sampai beberapa bulan, dengan preferensi kepada jangka waktu yang lebih pendek.
4. Rekan bisnis dan pengembang perangkat lunak harus bekerja-sama tiap hari sepanjang proyek.
5. Kembangkan proyek di sekitar individual yang termotivasi. Berikan mereka lingkungan dan dukungan yang mereka butuhkan, dan percayai mereka untuk menyelesaikan pekerjaan dengan baik.
6. Metode yang paling efisien dan efektif untuk menyampaikan informasi dari dan dalam tim pengembangan perangkat lunak adalah dengan komunikasi secara langsung.
7. Perangkat lunak yang bekerja adalah ukuran utama kemajuan.
8. Proses *Agile* menggalakkan pengembangan berkelanjutan. Sponsor-sponsor, pengembang-pengembang, dan pengguna-pengguna akan dapat mempertahankan kecepatan tetap secara berkelanjutan.

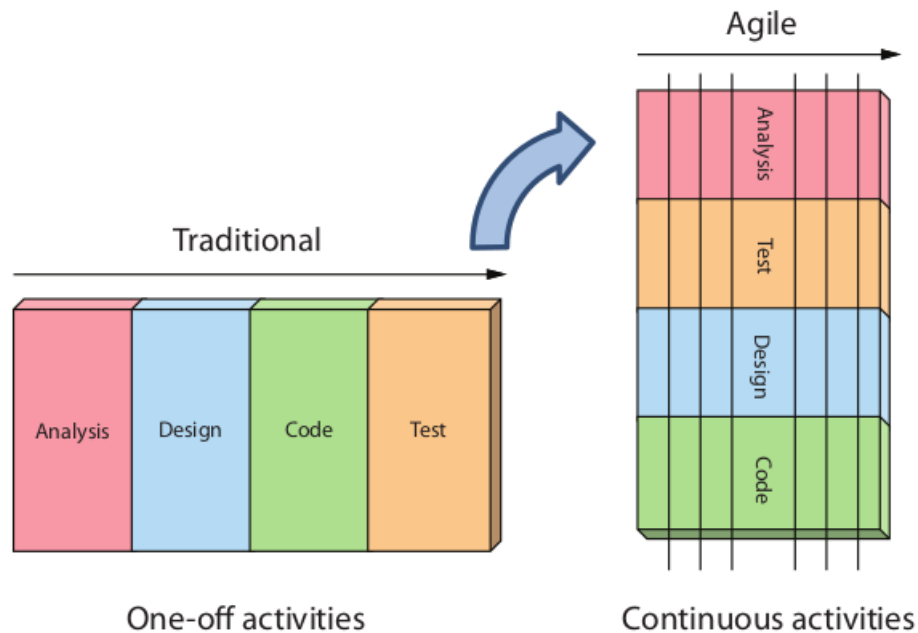
9. Perhatian yang berkesinambungan terhadap keunggulan teknis dan rancangan yang baik meningkatkan *Agility*.
10. Kesederhanaan—seni memaksimalkan jumlah pekerjaan yang belum dilakukan—adalah hal yang amat penting.
11. Arsitektur, kebutuhan, dan rancangan perangkat lunak terbaik muncul dari tim yang dapat mengorganisir diri sendiri.
12. Secara berkala, tim pengembang berefleksi tentang bagaimana untuk menjadi lebih efektif, kemudian menyesuaikan dan menyelaraskan kebiasaan bekerja mereka.

#### **2.6.4 Bagaimana Agile Berbeda**

Sebenarnya, satu bab laporan ini tidak akan cukup untuk membahas bagaimana *Agile* berbeda dengan metodologi pengembangan perangkat lunak lainnya dikarenakan cakupan bahasannya yang luas, tetapi ada beberapa hal umum yang perlu diketahui tentang proyek-proyek pengembangan perangkat lunak yang menggunakan *Agile*.

Pertama, peran-peran yang diterapkan di proyek-proyek *Agile* lebih fleksibel. Ketika diterapkan dengan tepat, bergabung dengan tim *Agile* seperti di dalam sebuah *mini-startup* (alias *startup company*). Orang-orang di dalam tim bekerja dengan penuh semangat dan mengerjakan apa yang bisa dilakukan untuk membuat proyek menjadi sukses tanpa memperhatikan jabatan atau peran yang diemban.

Kedua, yang membuat *Agile* berbeda dengan yang lain adalah proses analisis, desain, *coding*, dan pengujian merupakan aktivitas yang kontinu. Itu berarti kegiatan ini tidak bisa berlangsung dalam sifat yang terisolasi lagi. Ilustrasi aktivitas yang kontinu tersebut bisa dilihat pada Gambar 2.7.



Gambar 2.7: Ilustrasi aktivitas kontinu di *Agile* [21]

### 2.6.5 Beberapa Praktek Agile yang Diterapkan

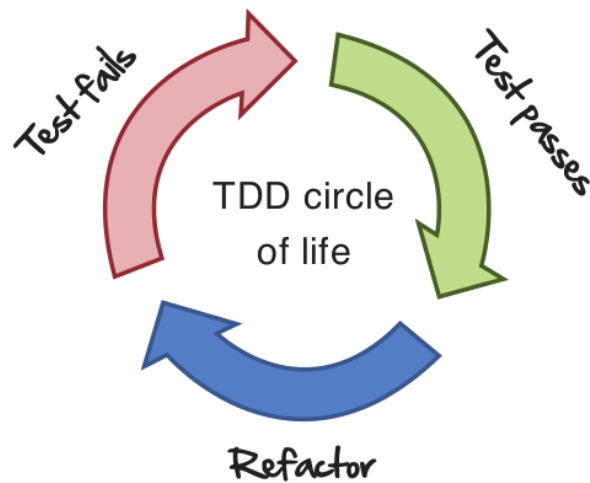
Mengingat pengetahuan penulis tentang Agile masih jauh dari cukup, cakupan pengerjaan dari purwa-rupa cukup kecil dan tim pengerjaan teknis yang hanya terdiri dari satu orang, yaitu penulis sendiri. Maka hanya sebagian kecil praktek *Agile* yang diterapkan oleh penulis, yaitu:

1. Mencatat daftar fitur dalam bentuk *user story*
2. Menerapkan jangka waktu 1 minggu untuk setiap iterasi
3. Melakukan *sprint planning* untuk merencanakan daftar tugas yang perlu dikerjakan selama iterasi berlangsung

### 2.6.6 Mempraktekkan Test-Driven Development

Untuk mengembangkan purwa-rupa Web API, penulis menggunakan teknik pengembangan perangkat lunak *test-driven development* (TDD). TDD adalah teknik pengembangan perangkat lunak yang menggunakan siklus pengembangan yang sangat singkat

untuk merancang perangkat lunak secara bertahap. Ilustrasi dari siklus pengembangan yang dimaksud dapat dilihat pada gambar 2.8.



Gambar 2.8: Bagaimana TDD Bekerja [21]

Berikut adalah penjelasan detail dari masing-masing tahap dari gambar 2.8:

1. **Test Fails.** Sebelum menulis kode baru untuk sistem, hal pertama yang harus dilakukan adalah menulis *unit test* yang gagal, untuk menunjukkan maksud dari apa yang diinginkan<sup>22</sup> dari kode baru yang akan ditulis. Di sini pengembang aplikasi berpikir kritis tentang desain.
2. **Test Passes.** Kemudian pengembang aplikasi melakukan apapun yang diperlukan untuk membuat tes lulus.
3. **Refactor.** Kemudian kembali menyentuh kode yang sudah ditulis, untuk membersihkan kode atau "dosa" yang sudah dilakukan ketika mencoba untuk mendapatkan lulus tes. Di sini akan dihapus duplikasi dan memastikan semuanya ramping, berarti, dan sejelas mungkin.

Dengan begini kita hanya mewujudkan apa yang perlu dalam bentuk tes. Membuat kode dengan cara ini memastikan pengembang aplikasi hanya membuat yang dibutuhkan dan fokus. Tes hanya digunakan sebagai pintu masuk untuk mewujudkan niat pengembang aplikasi. Hal ini yang menyebabkan mengapa TDD sering dimaksud sebagai teknik desain, dan sedikit tentang pengujian [21]. Jika ingin mengetahui lebih banyak tentang TDD, silahkan untuk membaca referensi berikut:

1. *Test-driven Development* dari Wikipedia<sup>23</sup>

<sup>22</sup>Arti dari "menunjukkan maksud dari apa yang diinginkan" adalah **merancang**.

<sup>23</sup>[http://en.wikipedia.org/wiki/Test-driven\\_development](http://en.wikipedia.org/wiki/Test-driven_development)

2. *Agile Best Practice: Executable Specifications*<sup>24</sup>

3. *Introduction to Test Driven Development (TDD)* dari Agile Data<sup>25</sup>

## 2.6.7 Hasil Akhir Implementasi

Sesuai dengan *milestone* yang sudah ditentukan di awal, hasil akhir dari implementasi Web API yang penulis lakukan adalah:

1. Web API sudah mampu untuk merespon permintaan konten detail event dalam format JSON melalui URL <http://ghanozjson.ap01.aws.af.cm/event/:id>.
2. Web API sudah mampu untuk merespon permintaan konten detail event dalam format XML melalui URL <http://ghanozjson.ap01.aws.af.cm/event/:id>
3. Web API sudah mampu untuk merespon permintaan daftar event dalam format JSON melalui URL <http://ghanozjson.ap01.aws.af.cm/events>.
4. Web API sudah mampu untuk merespon permintaan daftar event dalam format XML melalui URL <http://ghanozjson.ap01.aws.af.cm/events>.
5. Data yang dihasilkan dari API daftar event hanya menampilkan maksimal sebanyak 15 item event dalam sekali permintaan.

Masing-masing detail respon API tersebut dapat dilihat pada lampiran C.

---

<sup>24</sup><http://www.agilemodeling.com/essays/executableSpecifications.htm>

<sup>25</sup><http://www.agiledata.org/essays/tdd.html>

## Bab 3

# Analisis Sistem

### 3.1 Roadmap Pengerjaan Purwa-rupa Web API

Sebelum penulis memulai *software development life cycle*, penulis terlebih dahulu menentukan *roadmap* dari pengembangan Web API yang terdiri dari 3 *milestone* yang harus dipenuhi. Berikut daftar *milestone* tersebut:

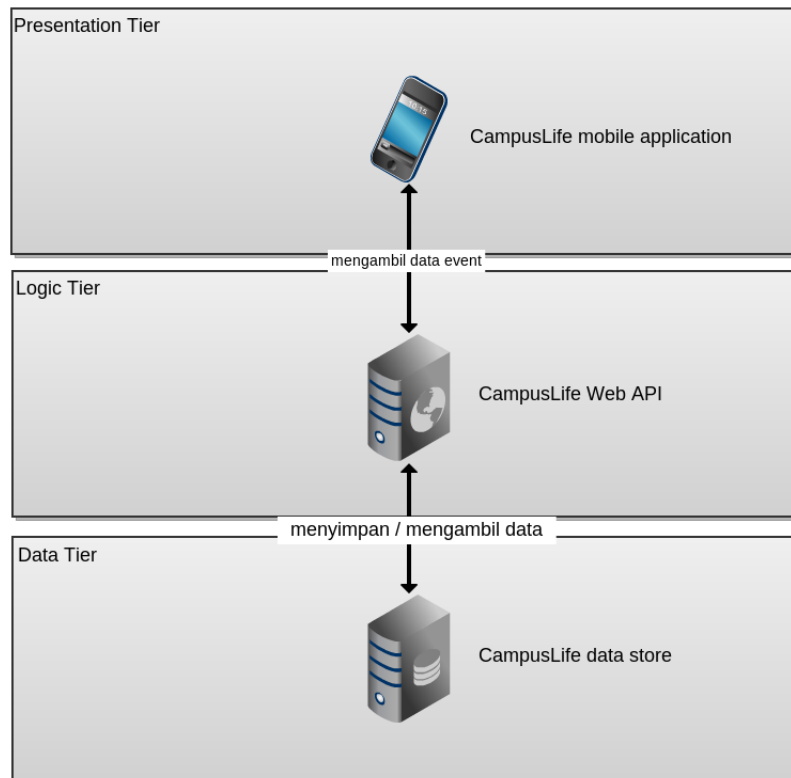
1. M 1: *Application Skeleton*. Pada *milestone* ini penulis mempersiapkan struktur aplikasi<sup>1</sup>
2. M 2: *Make it works and right*. Pada *milestone* ini penulis mengimplementasikan Web API sesuai dengan kebutuhan dan rancangan yang sudah ditentukan.
  - (a) API untuk merespon permintaan konten event dalam format JSON
  - (b) API untuk merespon permintaan konten event dalam format XML
3. M 3: *Measuring*
  - (a) Mengukur jumlah permintaan yang berhasil dilakukan dalam 90 detik ketika meminta konten event dalam format XML
  - (b) Mengukur jumlah permintaan yang berhasil dilakukan dalam 90 detik ketika meminta konten event dalam format JSON
  - (c) Membuktikan bahwa data yang dihasilkan dalam format JSON lebih kecil ukurannya dibandingkan data yang dihasilkan dalam format XML

---

<sup>1</sup>[http://en.wikipedia.org/wiki/Skeleton\\_\(computer\\_programming\)](http://en.wikipedia.org/wiki/Skeleton_(computer_programming))

## 3.2 High-level Architecture CampusLife

Secara umum CampusLife dibangun dengan menggunakan *three-tier architecture* yang terdiri dari 3 tingkat, yaitu *presentation*, *logic* dan *data*. CampusLife dibangun dengan menggunakan *three-tier architecture* dengan tujuan agar pengembang aplikasi bisa melakukan perubahan dengan relatif mudah. Gambaran visual dari arsitektur CampusLife dapat dilihat pada gambar 3.1:



Gambar 3.1: High-level architecture CampusLife

### 3.2.1 Presentation Tier

*Presentation tier* adalah *tier* paling atas dari aplikasi CampusLife. *Tier* ini menampilkan informasi terkait dengan layanan seperti menampilkan daftar event dan menampilkan daftar pengunjung event kepada pengguna melalui *mobile application*. Layanan informasi seperti ini akan disediakan melalui Web API yang berada di *logic tier* dan dapat diakses melalui *client* seperti *mobile application* dan *browser*. Secara sederhana *presentation tier* adalah tingkat dimana pengguna dapat mengakses aplikasi secara langsung. Dalam konteks pengerjaan penelitian tugas akhir ini, *mobile application* CampusLife akan berada di *tier* ini [7].

### 3.2.2 Logic Tier

*Logic tier* adalah *tier* yang berfungsi untuk mengontrol fungsionalitas aplikasi dengan melakukan pengolahan rinci. Contoh pengolahan rinci yang bisa dilakukan di *tier* ini adalah mengambil daftar event. Dalam konteks pengerjaan penelitian tugas akhir ini, Web API akan berada di *tier* ini [7].

### 3.2.3 Data Tier

*Data tier* adalah *tier* yang terdiri dari satu atau lebih *server* basis data. Disini setiap informasi disimpan dan diambil. *Tier* ini akan menjaga data tetap netral dan independen dari *application server* atau *business logic*. Selain itu dengan memberikan data di *tier* sendiri akan meningkatkan *scalability* dan *performance*. Dalam konteks pengerjaan penelitian tugas akhir ini, penulis menggunakan MongoDB<sup>2</sup> sebagai basis data di *tier* ini [7].

### 3.2.4 Keuntungan dari Arsitektur Three-tier

CampusLife dibangun dengan menggunakan arsitektur *three-tier* untuk mendapatkan beberapa keuntungan berikut:

1. Lebih mudah untuk memodifikasi atau mengganti *tier* apapun tanpa mempengaruhi *tier* lainnya.
2. Memisahkan aplikasi dan fungsi *database* berarti *load balancing* yang lebih baik.

### 3.2.5 Ruang Lingkup Pengerjaan

Berdasarkan batasan masalah yang sudah penulis jelaskan di Bab 1, berikut adalah ruang lingkup pengerjaan yang penulis perlu dilakukan untuk membangun Web API:

1. Membangun Web API
2. Implementasi rancangan skema basis data di MongoDB

---

<sup>2</sup><http://www.mongodb.org/>



### 3.2.6 Daftar Kebutuhan yang harus dipenuhi

Sebelum Web API dibangun, penulis menentukan terlebih dahulu daftar kebutuhan fungsional yang ada di Web API. Berikut daftar kebutuhan fungsional dan non-fungsional yang harus ada di Web API:

1. Web API harus mampu untuk merespon permintaan daftar event dalam format JSON
2. Web API harus mampu untuk merespon permintaan daftar event dalam format XML
3. Web API harus mampu untuk merespon permintaan konten detail event dalam format JSON
4. Web API harus mampu untuk merespon permintaan konten detail event dalam format XML
5. Data yang dihasilkan Web API harus memiliki ukuran yang optimal sehingga aplikasi mobile dapat mengambil data dengan waktu yang relatif singkat

Dari kelima daftar kebutuhan di atas, daftar kebutuhan ke-5 merupakan kebutuhan non-fungsional dan sisanya merupakan daftar kebutuhan fungsional.

Mengingat penulis mempraktekkan *Agile* untuk membangun Web API ini, maka perlu diketahui dalam sehari-harinya setiap daftar kebutuhan fungsional dan non-fungsional tersebut biasa disebut dengan *User Story*<sup>3</sup> dan semua daftar *user story* tersebut ditempatkan ke dalam *Product Backlog*.

### 3.2.7 Pola Alur Kerja Ketika Mengumpulkan Daftar Kebutuhan

Ketika melakukan pengumpulan daftar kebutuhan penulis menemukan pola alur kerja sebagai berikut:

1. Membuat *user story* format singkat
2. Membuat *user story* format lengkap

---

<sup>3</sup>Lebih detail tentang *user story* silahkan baca di Bab 2 Landasan Teori

### 3.2.7.1 Membuat User Story format singkat

Pada prakteknya jika penulis sedang dalam keadaan yang terburu-buru, penulis biasanya mencatat fitur-fitur baru yang akan dikembangkan dengan format singkat. Berikut adalah contoh daftar fitur yang dibuat dengan *user story* format singkat:

1. *List upcoming events*
2. *Display upcoming competitions*
3. *Display event details*

Poin penting yang perlu diingat ketika membuat *user story* format singkat adalah:

1. Mencatat **apa yang harus dilakukan oleh perangkat lunak**
2. Mendeskripsikan fitur dengan **sesingkat dan sejelas mungkin**

Tidak ada yang tahu hari esok seperti apa, **jangan tergoda untuk mendeskripsikan sedetail mungkin** ketika membuat *user story* format singkat.

### 3.2.7.2 Membuat User Story format panjang

Pada prakteknya, penulis membuat *user story* format panjang setelah mendefinisikan fitur dalam *user story* format panjang. Berikut adalah format *user story* format panjang:

```
As a <type of user>
I want <some goal>
so that <some reason>
```

Berikut adalah format panjang yang diberi anotasi:

```
As a <type of user> => Who is this story for
I want <some goal> => What they want to do
so that <some reason> => Why they want to do it
```

Setelah *user story* format panjang dibuat, biasanya penulis meneruskan dengan menentukan *acceptance criteria* untuk masing-masing *user story* tersebut. *Acceptance criteria* tersebut akan digunakan saat melakukan *acceptance testing*.

### 3.2.8 Acceptance Criteria

*Acceptance criteria* merupakan salah *requirements artifact* yang akan berguna saat *acceptance testing* untuk menentukan apakah kebutuhan sudah terpenuhi atau belum [19]. Berikut adalah daftar masing-masing *acceptance criteria* untuk setiap masing-masing *user story*:

1. *Get Event Details Acceptance Criteria*

- (a) *Scenario 1: Event Details request should return response in JSON*
- (b) *Scenario 2: Event Details request should return response in XML*
- (c) *Scenario 3: Unsupported Content-Type should return HTTP 406 code*

2. *Get Event List Acceptance Criteria*

- (a) *Scenario 1: Event List request should return response in JSON*
- (b) *Scenario 2: Event List request should return response in XML*
- (c) *Scenario 3: Unsupported Content-Type should return HTTP 406 code*

Setiap masing-masing skenario dari *acceptance criteria* di atas diekspresikan dalam format *Given When Then* [22][23]. Detail masing-masing dari setiap *acceptance criteria* diatas bisa dilihat pada lampiran A.

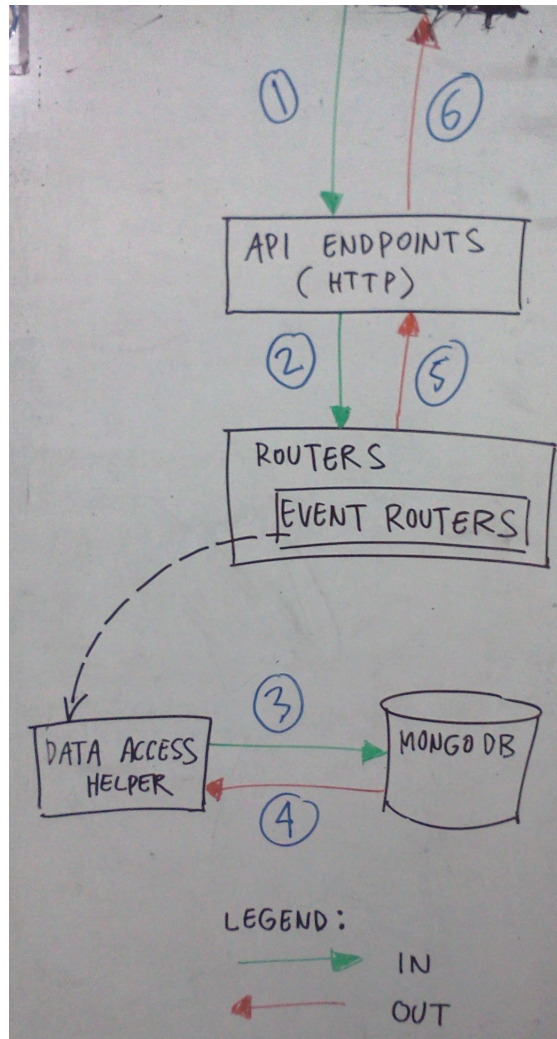
## Bab 4

# Perancangan Sistem

Sebelum membaca lebih lanjut bab ini, para pembaca perlu mengetahui bahwa saat melakukan pemodelan, penulis mempraktekkan *Just Barely Good Enough modelling*[24]. Jadi, jangan terkejut jika menemukan model yang diperlihatkan di bab ini mungkin tidak "benar" menurut perspektif pembaca karena saat itu model yang dibuat dianggap cukup berdasarkan pemahaman dan situasi yang dihadapi oleh penulis pada saat proses pemodelan berlangsung.

### 4.1 Arsitektur Teknis Web API

Untuk merancang arsitektur teknis dari Web API, penulis menggunakan *free-form diagram*. Model arsitektur teknis yang dibuat digambarkan pada papan tulis dan model tersebut dapat dilihat pada gambar 4.1



Gambar 4.1: Model Arsitektur Teknis Web API dalam Free-form Diagram

Dikarenakan penulis menggunakan *free-form diagram* untuk merancang arsitektur teknis dari Web API, berikut adalah keterangan dari beberapa bagian dari model di atas:

1. *HTTP Request* masuk dari *client* melalui salah satu *API endpoint*.
2. *HTTP Request* yang masuk akan diproses lebih lanjut oleh *Routers*
3. *Routers* akan memiliki *Data Access helper* untuk mengakses data mentah di MongoDB
4. *Routers* mendapatkan data untuk kemudian diolah sesuai dengan format yang diminta oleh *client*
5. *HTTP Response* dikirimkan kepada *client*

## 4.2 Service Interaction Model

Pada bagian ini akan berisi laporan mengenai hasil pemodelan yang penulis lakukan saat melakukan perancangan sistem. Untuk mendapatkan gambaran dengan jelas mengenai sistem yang akan dibangun, penulis melakukan pemodelan interaksi antara *client* dan Web API *server*. Model tersebut dibuat dalam bentuk *UML Sequence Diagram* dan digambarkan pada papan tulis. Namun, dalam laporan ini penulis memperlihatkan hasil pemodelan yang sudah dirubah ke dalam bentuk digital.

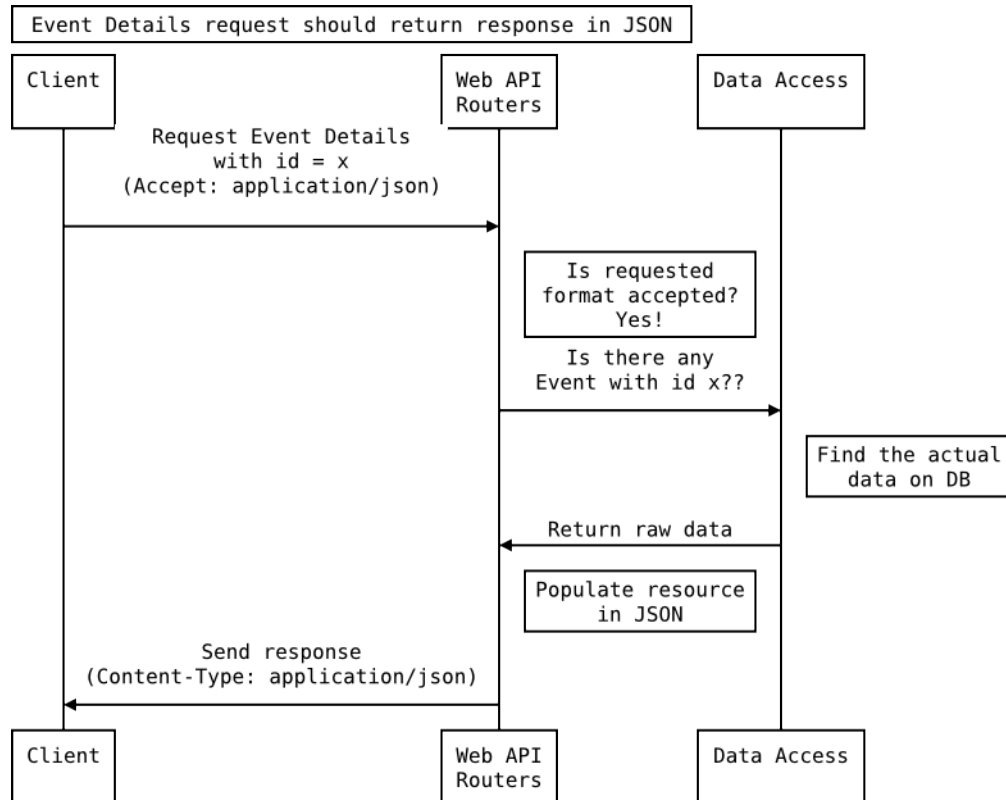
Saya asumsikan para pembaca sudah mengetahui penggunaan dasar dari *UML Sequence Diagram*, jika belum mohon untuk memahami terlebih dahulu penggunaan dasar *UML Sequence Diagram*<sup>1</sup>.

---

<sup>1</sup><http://www.agilemodeling.com/artifacts/sequenceDiagram.htm>

### 4.2.1 Permintaan Detail Event dalam format JSON

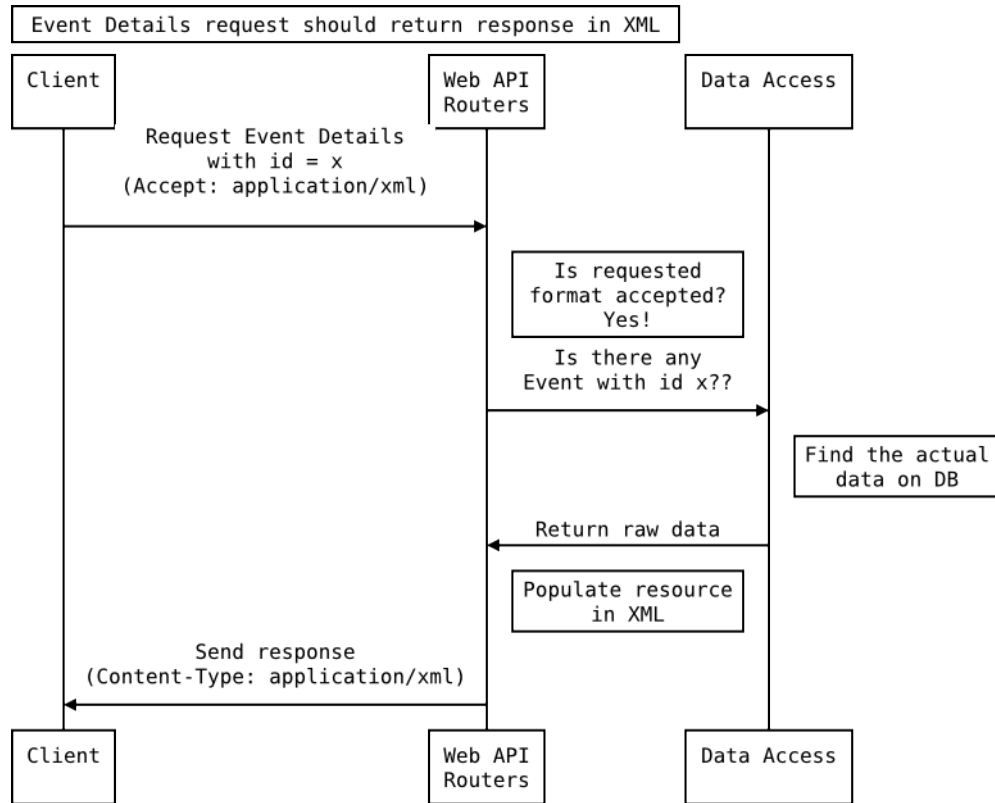
Pemodelan interaksi antara *client* dan Web API *server* saat *client* meminta detail event dalam format JSON dapat dilihat pada gambar 4.2.



Gambar 4.2: Permintaan Detail Event dalam Format JSON

### 4.2.2 Permintaan Detail Event dalam format XML

Pemodelan interaksi antara *client* dan Web API *server* saat *client* meminta detail event dalam format XML dapat dilihat pada gambar 4.3.

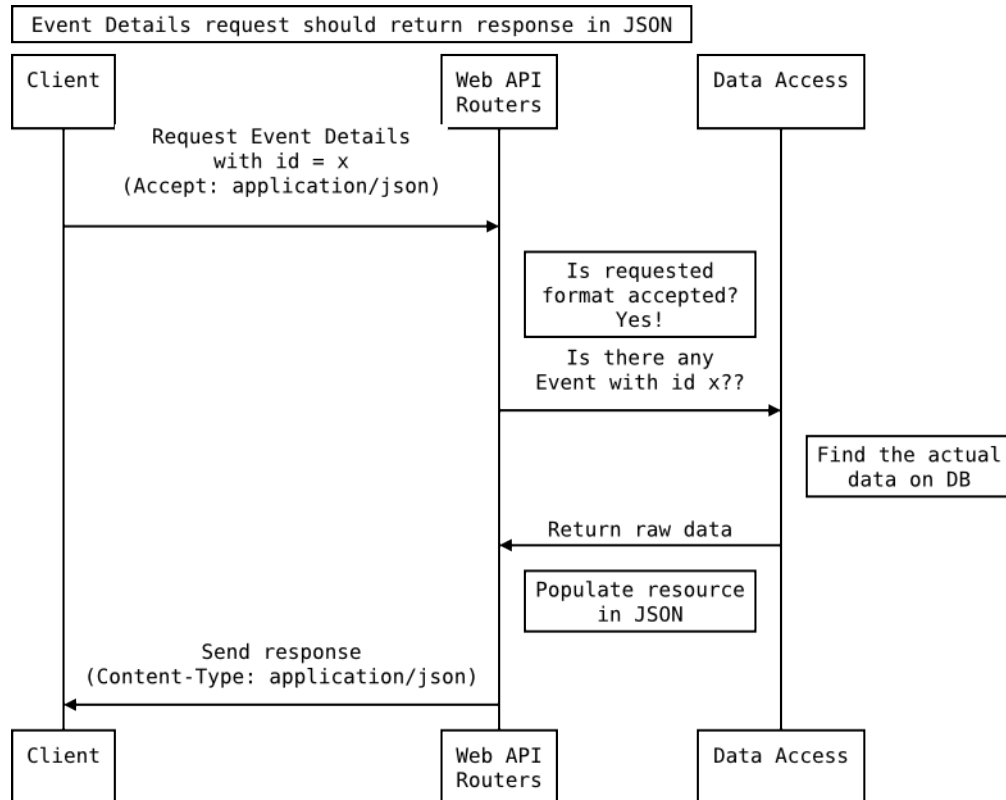


Gambar 4.3: Permintaan Detail Event dalam Format XML



### 4.2.3 Permintaan Daftar Event dalam format JSON

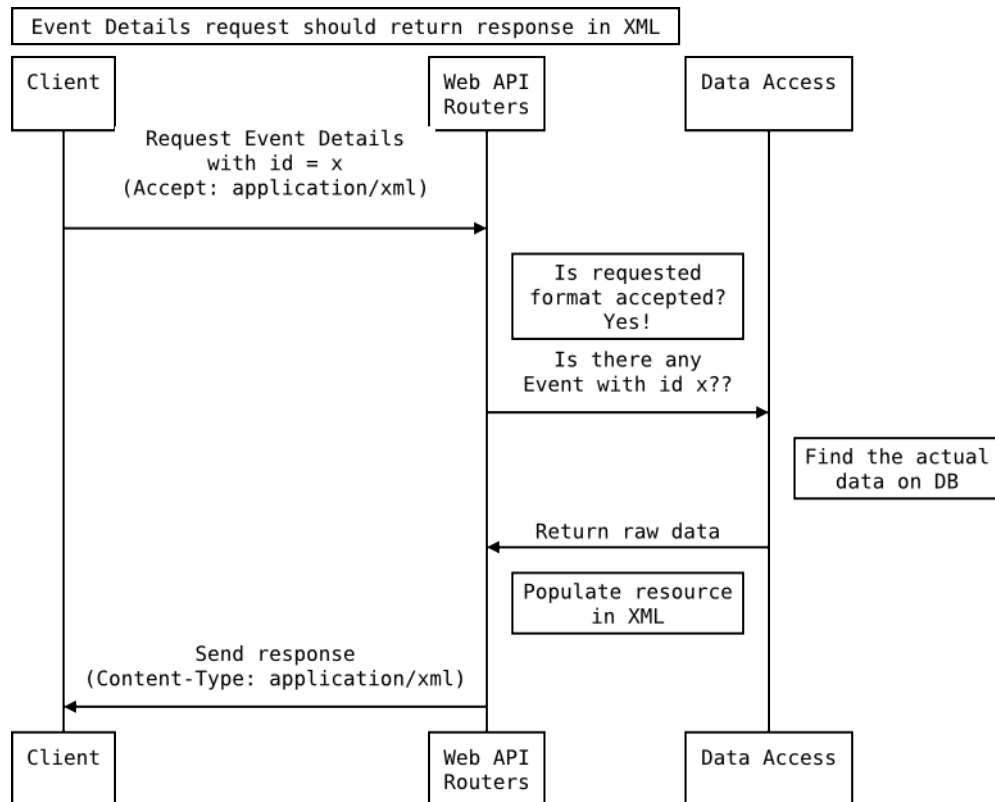
Pemodelan interaksi antara *client* dan Web API server saat *client* meminta daftar event dalam format JSON dapat dilihat pada gambar 4.4.



Gambar 4.4: Permintaan Daftar Event dalam Format JSON

#### 4.2.4 Permintaan Daftar Event dalam format XML

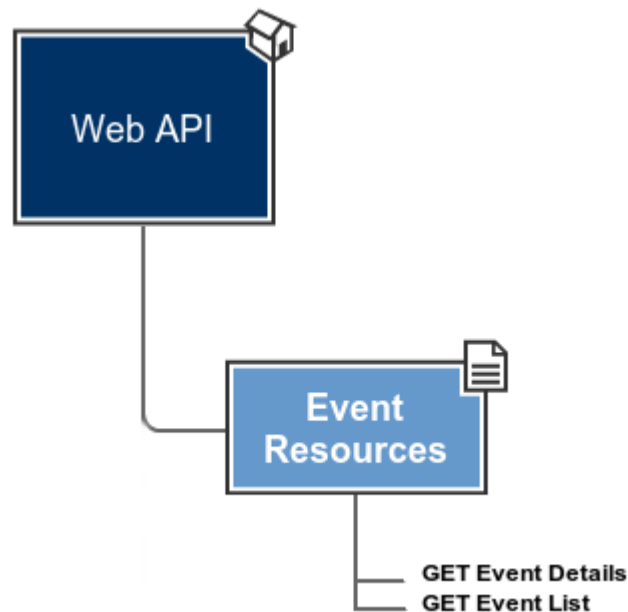
Pemodelan interaksi antara *client* dan Web API *server* saat *client* meminta daftar event dalam format XML dapat dilihat pada gambar 4.5.



Gambar 4.5: Permintaan Daftar Event dalam Format XML

## 4.3 Struktur URI

Web API diakses melalui *HTTP endpoints*. *HTTP endpoints* tersebut merupakan struktur URI (*Uniform Resource Identifier*) yang digunakan untuk mengakses *resources*. Representasi visual dalam bentuk *Site Diagram* dari struktur URI tersebut bisa dilihat pada gambar 4.6



Gambar 4.6: Struktur URI Web API

## 4.4 Rancangan Skema Representasi Resource

### 4.4.1 Pertimbangan Rancangan Skema Representasi Resource dalam format JSON

#### 4.4.1.1 Rancangan Skema Representasi Resource Detail Event dalam format JSON

Berikut adalah beberapa pertimbangan desain untuk rancangan skema representasi *resource* detail event:

1. Setiap *resource* detail event akan memiliki URL dalam konteks Web API untuk memberitahukan dimana lokasi *resource* yang sedang diakses
2. Data aktual detail event akan dibungkus dalam propertie data yang berupa *object*

Detail rancangan skema dapat dilihat pada lampiran B.

#### **4.4.1.2 Rancangan Skema Representasi Resource Daftar Event dalam format JSON**

Berikut adalah beberapa pertimbangan desain untuk rancangan skema representasi *resource* daftar event:

1. Setiap daftar event resource akan disertai properti tambahan url untuk memberitahukan dimana lokasi resource yang sedang diakses
2. Data aktual Event List akan dibungkus dalam variable data yang berupa array

Detail rancangan skema dapat dilihat pada lampiran B.

#### **4.4.1.3 Rancangan Skema Representasi Resource Detail Event dalam format XML**

Berikut adalah beberapa pertimbangan desain untuk rancangan skema representasi *resource* detail event:

1. Setiap *resource* detail event akan memiliki URL dalam konteks Web API untuk memberitahukan dimana lokasi *resource* yang sedang diakses
2. Data aktual detail event akan dibungkus dalam elemen data

Detail rancangan skema dapat dilihat pada lampiran B.

#### **4.4.1.4 Rancangan Skema Representasi Resource Daftar Event dalam format XML**

Berikut adalah beberapa pertimbangan desain untuk rancangan skema representasi *resource* daftar event:

1. Setiap *resource* daftar event akan memiliki URL dalam konteks Web API untuk memberitahukan dimana lokasi *resource* yang sedang diakses
2. Data aktual daftar event akan dibungkus dalam elemen data. Setiap butir event akan dibungkus dalam elemen data yang disertai atribut id.
3. Setiap *resource* daftar event akan disertai properti URL

Detail rancangan skema dapat dilihat pada lampiran B.

## Bab 5

# Implementasi dan Pengujian Sistem

Pada bagian ini akan dibahas mengenai apa saja yang sudah diimplementasikan dan teknologi berkaitan yang digunakan untuk membangun purwa-rupa Web API.

### 5.1 Implementasi Web API

#### 5.1.1 Teknologi yang Dipakai untuk Membangun Web API

Untuk membangun purwa-rupa Web API ini penulis menggunakan beberapa teknologi utama berikut:

1. Node.js<sup>1</sup>. Node.js adalah *platform* yang dibangun di atas Chrome JavaScript *runtime*<sup>2</sup> untuk dengan mudah membangun aplikasi jaringan yang cepat dan *scalable*. Penulis menggunakan Node.JS sebagai *platform* dari Web API.
2. Express<sup>3</sup>. Express adalah *framework* aplikasi web minimal dan fleksibel, menyediakan satu set fitur yang kuat untuk membangun aplikasi web tunggal dan multi-halaman, dan *hybrid*.
3. MongoDB<sup>4</sup>. Basis-data berorientasi dokumen *open-source*<sup>5</sup>, dan basis data NoSQL<sup>6</sup> terkemuka.

---

<sup>1</sup><http://nodejs.org/>

<sup>2</sup><https://code.google.com/p/v8/>

<sup>3</sup><http://expressjs.com/>

<sup>4</sup><http://www.mongodb.org/>

<sup>5</sup><http://docs.mongodb.org/manual/core/data-modeling/>

<sup>6</sup><https://en.wikipedia.org/wiki/NoSQL>

## 5.2 Pengujian Web API

Penulis menerapkan satu level pengujian perangkat lunak untuk melakukan pengujian Web API, yaitu dengan melakukan *acceptance testing*. Keseluruhan *test suite* bisa dilihat di repositori *ghanoz-json*<sup>7</sup>.

### 5.2.1 Skenario Pengujian

Pengujian secara umum terbagi ke dalam dua skenario besar, yaitu skenario pengujian untuk mengambil detail event dan daftar event. Keluaran log dari hasil pengujian tersebut dapat dilihat pada lampiran D.

#### 5.2.1.1 Skenario Pengujian untuk Mengambil Detail Event

Untuk menguji perangkat lunak di level *acceptance*, berikut adalah daftar skenario yang diterapkan ketika melakukan pengujian:

1. *Scenario 1: Event Details request should return response in JSON*
2. *Scenario 2: Event Details request should return response in XML*
3. *Scenario 3: Unsupported Content-Type should return HTTP 406 code*

#### 5.2.1.2 Skenario Pengujian untuk Mengambil Daftar Event

Untuk menguji perangkat lunak di level *acceptance*, berikut adalah daftar skenario yang diterapkan ketika melakukan pengujian:

1. *Scenario 1: Event List request should return response in JSON*
2. *Scenario 2: Event List request should return response in XML*
3. *Scenario 3: Unsupported Content-Type should return HTTP 406 code*

## 5.3 Hasil Pembandingan Ukuran Data JSON dan XML

Penulis juga melakukan pembandingan antara JSON dan XML dalam hal ukuran data yang dihasilkan dan jumlah respon permintaan yang berhasil dikembalikan. Pembandingan akan dilakukan dengan menggunakan data yang sama dan jumlah detik yang digunakan saat melakukan pembandingan.

---

<sup>7</sup><https://github.com/muhammadghazali/ghanoz-json/tree/master/test/api-end-point-test>

Untuk melakukan hal ini penulis menggunakan Apache Benchmark<sup>8</sup>. Kakas tersebut akan bertugas mengirimkan permintaan Web API *end point* daftar event. Berikut adalah skenario yang penulis lakukan:

1. *Empty Resource*. Melakukan perbandingan jumlah permintaan yang berhasil direspon dengan respon yang dikembalikan berisi *resource* dengan setiap *field* dari *resource* diberi *resource* kosong dengan setiap *field* dari *resource* diberi *string* kosong
2. *Normal Resource*. Melakukan perbandingan jumlah permintaan yang berhasil direspon dengan respon yang dikembalikan berisi *resource* dengan setiap *field* dari *resource* diberi data yang "normal"
3. *Just Once*. Melakukan perbandingan ukuran data terhadap hasil respon yang dikembalikan berisi *resource* dengan setiap *field* dari *resource* diberi data yang "normal"

Detail masing-masing hasil perbandingan tersebut ada di lampiran E

---

<sup>8</sup><http://httpd.apache.org/docs/2.4/programs/ab.html>

# Bab 6

## Penutup

### 6.1 Kesimpulan

Selama menjalani masa pengerjaan tugas akhir penulis mendapatkan beberapa kesimpulan dari hasil penerapan JSON sebagai format serialisasi data:

1. Berdasarkan hasil pengujian yang sudah dilakukan, terbukti bahwa data yang diserialisasi dalam format JSON memiliki ukuran data yang lebih kecil<sup>1</sup> jika dibandingkan dengan data yang diserialisasi dalam format XML.
2. Implikasi dari penerapan JSON pada *resource* adalah jumlah permintaan yang berhasil dikembalikan jauh lebih banyak<sup>2</sup> dibandingkan ketika Web API mengembalikan *resource* dalam format XML.

### 6.2 Saran

Jika tertarik untuk menerapkan JSON untuk serialisasi data, penulis menyarankan beberapa tips berikut:

1. Gunakan nama *field* yang sejelas dan ringkas mungkin.
2. JSON tepat jika digunakan untuk menserialisasikan data yang berbasis teks.

Demikian saran-saran yang dapat penulis sampaikan, penulis berharap laporan hasil pengerjaan tugas akhir ini dapat bermanfaat bagi semua pihak yang membaca.

---

<sup>1</sup>Lihat Lampiran E di bagian "*Just Once*"

<sup>2</sup>Lihat Lampiran E di bagian "*Empty Resource*" dan "*Normal Resource*"



# Bibliografi

- [1] Deepak, G., and Dr. Pradeep B S. "Challenging Issues and Limitations of Mobile Computing." *International Journal of Computer Technology and Applications* 3.1 (2012): Academic Journals Database. Web. 8 Jan. 2013.
- [2] Audie Sumaray dan S. Kami Makki. "A comparison of data serialization formats for optimal efficiency on a mobile platform". *6th International Conference on Ubiquitous Information Management and Communication* (2012): Artikel No. 48. ACM Digital Library. Web. 24 Jan 2013.
- [3] *Debate: JSON vs. XML as a data interchange format* <http://www.infoq.com/news/2006/12/json-vs-xml-debate> diakses pada 20 Januari 2012.
- [4] *Serialization* <http://en.wikipedia.org/wiki/Serialization> diakses pada 24 Januari 2012.
- [5] *Agile software development* [http://en.wikipedia.org/wiki/Agile\\_software\\_development](http://en.wikipedia.org/wiki/Agile_software_development) diakses pada 24 Januari 2012.
- [6] *JSON: The Fat-Free Alternative to XML* <http://www.json.org/xml.html> diakses pada 20 Januari 2012.
- [7] *Wikipedia: Multitier architecture* [http://en.wikipedia.org/wiki/Multitier\\_architecture](http://en.wikipedia.org/wiki/Multitier_architecture) diakses pada 15 Maret 2013.
- [8] *Three-Tier Architecture* <http://www.linuxjournal.com/article/3508> diakses pada 18 Maret 2013.
- [9] *Application Programming Interface* [http://en.wikipedia.org/wiki/Application\\_programming\\_interface](http://en.wikipedia.org/wiki/Application_programming_interface) diakses pada 22 Maret 2013.
- [10] *Web API* [http://en.wikipedia.org/wiki/Web\\_API](http://en.wikipedia.org/wiki/Web_API) diakses pada 20 Januari 2013.
- [11] *APIs* <http://www.linuxjournal.com/content/apis> diakses pada 20 Januari 2013.
- [12] *RESTful Web services: The basics* <http://www.ibm.com/developerworks/webservices/library/ws-restful/> diakses pada 14 September 2012.
- [13] *Introducing JSON* <http://www.json.org/> diakses pada 20 Januari 2013.

- [14] *The application/json Media Type for JavaScript Object Notation (JSON)*  
<http://tools.ietf.org/html/rfc4627> diakses pada 5 April 2013.
- [15] *How REST replaced SOAP on the Web: What it means to you*  
<http://www.infoq.com/articles/rest-soap> diakses pada 14 September 2012.
- [16] *XML* <http://en.wikipedia.org/wiki/XML> diakses pada 7 April 2013.
- [17] *JSON Continues its Winning Streak Over XML*  
<http://blog.programmableweb.com/2010/12/03/json-continues-its-winning-streak-over-xml/> diakses pada 8 April 2013.
- [18] *Why JSON will continue to push XML out of the picture*  
<http://blog.appfog.com/why-json-will-continue-to-push-xml-out-of-the-picture/> diakses pada 8 April 2013.
- [19] *Acceptance Testing* [http://en.wikipedia.org/wiki/Acceptance\\_testing](http://en.wikipedia.org/wiki/Acceptance_testing) diakses pada 7 Mei 2013.
- [20] *Manifesto Pengembangan Perangkat Lunak Agile*  
<http://agilemanifesto.org/iso/id/> diakses pada 8 April 2013.
- [21] *The Agile Samuuri* buku teks
- [22] *Introducing BDD* <http://dannorth.net/introducing-bdd/> diakses pada 7 Mei 2013.
- [23] *What's in a Story?* <http://dannorth.net/whats-in-a-story/> diakses pada 7 Mei 2013.
- [24] *"Just Barely Good Enough" Models and Documents: An Agile Best Practice*  
<http://www.agilemodeling.com/essays/barelyGoodEnough.html> diakses pada 10 Mei 2013.

# Lampiran A

## Acceptance Criteria

### A.1 Get Event Details Acceptance Criteria

#### A.1.1 Scenario 1: Event Details request should return response in JSON

*Given Event Details resource is accessible*

*And Event Details resource is accessible*

*When the client request the Event Details in JSON*

*Then the Web API should return Event Details in JSON*

*And returned response is supplied with the Event Details resource URL*

#### A.1.2 Scenario 2: Event Details request should return response in XML

*Given Event Details resource is accessible*

*And Event Details resource is accessible*

*When the client request the Event Details in XML*

*Then the Web API should return Event Details in XML*

*And returned response is supplied with the Event Details resource URL*

### **A.1.3 Scenario 3: Unsupported Content-Type should return HTTP 406 code**

*Given the Web API only support JSON and XML for the response*

*When the client request the resource with Content-Type other than JSON or XML*

*Then the Web API should return HTTP 406 code*

*And returned response should be contained an client error message*

## **A.2 Get Event List Acceptance Criteria**

### **A.2.1 Scenario 1: Event List request should return response in JSON**

*Given Event List resource is accessible*

*And Event List resource is accessible*

*When the client request the Event List in JSON*

*Then the Web API should return Event List in JSON*

*And returned response is should be accompanied by properties like resource url, page, pages, pageSize, total*

*And the maximum of number of item on the Event List is 15 Event*

### **A.2.2 Scenario 2: Event List request should return response in XML**

*Given Event List resource is accessible*

*And Event List resource is accessible*

*When the client request the Event List in XML*

*Then the Web API should return Event List in XML*

*And returned response is should be accompanied by properties like resource url, page, pages, pageSize, total*

*And the maximum of number of item on the Event List is 15 Event*

### **A.2.3 Scenario 3: Unsupported Content-Type should return HTTP 406 code**

*Given the Web API only support JSON and XML for the response*

*When the client request the resource with Content-Type other than JSON or XML*

*Then the Web API should return HTTP 406 code*

*And returned reponse should be contained an client error message*

## Lampiran B

### Detail Rancangan Skema Representasi Resource

#### B.1 Rancangan Skema Representasi Resource Detail Event

##### B.1.1 Rancangan Skema Representasi Resource Detail Event dalam format JSON

Tanpa menghiraukan nilai dari masing-masing *field*, berikut adalah rancangan skema representasi *resource* detail event dalam format JSON:

```
{
  "url": value ,
  "data": {
    "title": value ,
    "shortDescription": value ,
    "times": [
      {
        "milliseconds": value ,
        "fullFormat": value ,
        "year": value ,
        "month": value ,
        "date": value ,
        "day": value
      }
    ],
  },
}
```

```

        "isPassed": value ,
        "rating": value ,
        "lastUpdated": value ,
        "timestamp": value ,
        "_id": "value"
    }
}

```

Untuk melihat detail dari setiap contoh nilai setiap *field* dari rancangan skema di atas, silahkan untuk melihatnya di halaman web *ghanoz-json: Event Details resource representations design in JSON*<sup>1</sup>

### B.1.2 Rancangan Skema Representasi Resource Detail Event dalam format XML

Tanpa menghiraukan nilai dari masing-masing *field*, berikut adalah rancangan skema representasi *resource* detail event dalam format XML:

```

<response>
  <url>value</url>
  <data id="1">
    <title>value</title>
    <shortDescription>value</shortDescription>
    <times>
      <time>
        <milliseconds>value</milliseconds>
        <fullFormat>value</fullFormat>
        <year>value</year>
        <month>value</month>
        <date>value</date>
        <day>value</day>
      </time>
    </times>
    <isPassed>value</isPassed>
    <rating>value</rating>
    <lastUpdated>value</lastUpdated>
    <timestamp>value</timestamp>
  </data>
</response>

```

<sup>1</sup><https://gist.github.com/muhammadghazali/5409322#file-event-details-json>

```
</data>
</response>
```

Untuk melihat detail dari setiap contoh nilai setiap *field* dari rancangan skema di atas, silahkan untuk melihatnya di halaman web *ghanoz-json: Event Details resource representations design in XML*<sup>2</sup>

## B.2 Rancangan Skema Representasi Resource Daftar Event

Tanpa menghiraukan nilai dari masing-masing *field*, berikut adalah rancangan skema representasi *resource* daftar event dalam format JSON:

```
{
  "url": value ,
  "data": [
    {
      "title": value ,
      "shortDescription": value ,
      "times": [
        {
          "milliseconds": value ,
          "fullFormat": value ,
          "year": value ,
          "month": value ,
          "date": value ,
          "day": value
        }
      ],
      "isPassed": value ,
      "rating": value ,
      "lastUpdated": value ,
      "timestamp": value ,
      "_id": value
    },
    {
      "title": value ,
```

<sup>2</sup><https://gist.github.com/muhammadghazali/5409508#file-event-details-xml>



```

    "shortDescription": value ,
    "times": [
      {
        "milliseconds": value ,
        "fullFormat": value ,
        "year": value ,
        "month": value ,
        "date": value ,
        "day": value
      }
    ],
    "isPassed": value ,
    "rating": value ,
    "lastUpdated": value ,
    "timestamp": value ,
    "_id": value
  }
]
}

```

Untuk melihat detail dari setiap contoh nilai setiap *field* dari rancangan skema di atas, silahkan untuk melihatnya di halaman web *ghanoz-json: Event List resource representations design in JSON*<sup>3</sup>

### B.2.1 Rancangan Skema Representasi Resource Daftar Event dalam format XML

Tanpa menghiraukan nilai dari masing-masing *field*, berikut adalah rancangan skema representasi *resource* daftar event dalam format XML:

```

<response>
  <url>value </url>
  <data>
    <data id="value">
      <title>value </title>
      <shortDescription>value </shortDescription>
      <times>

```

<sup>3</sup><https://gist.github.com/muhammadghazali/5409417#file-event-list-json>

```

        <time>
            <milliseconds>value </milliseconds>
            <fullFormat>value </fullFormat>
            <year>value </year>
            <month>value </month>
            <date>value </date>
            <day>value </day>
        </time>
    </times>
    <isPassed>value </isPassed>
    <rating>value </rating>
    <lastUpdated>value </lastUpdated>
    <timestamp>value </timestamp>
</data>
<data id="value">
    <title>value </title>
    <shortDescription>value </shortDescription>
    <times>
        <time>
            <milliseconds>value </milliseconds>
            <fullFormat>value </fullFormat>
            <year>value </year>
            <month>value </month>
            <date>value </date>
            <day>value </day>
        </time>
    </times>
    <isPassed>value </isPassed>
    <rating>value </rating>
    <lastUpdated>value </lastUpdated>
    <timestamp>value </timestamp>
</data>
</data>
</response>

```

Untuk melihat detail dari setiap contoh nilai setiap *field* dari rancangan skema di atas, silahkan untuk melihatnya di halaman web *ghanoz-json: Event List resource*

*representations design in XML*<sup>4</sup>

---

<sup>4</sup><https://gist.github.com/muhammadghazali/5409528#file-event-list-xml>

# Lampiran C

## Detail Respon Web API

Perlu diingat bahwa beberapa detail di bagian ini sudah dirapihkan untuk keperluan penampilan di laporan ini.

### C.1 Hasil Respon Permintaan Detail Event

#### C.1.1 Hasil Respon Permintaan Detail Event dalam Format JSON

Berikut detail *request header* ketika melakukan permintaan detail event dalam format JSON:

```
Accept: application/json
User-Agent: Mozilla/5.0 (X11; Linux i686)
  AppleWebKit/537.31 (KHTML, like Gecko)
  Chrome/26.0.1410.63 Safari/537.31
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3
```

Berikut detail *response header* permintaan detail event yang dibalas oleh Web API:

```
Accept-Ranges: bytes
Age: 0
Content-Type: application/json
Date: Mon, 22 Apr 2013 19:01:40 GMT
Server: nginx
Vary: Accept
Via: 1.1 varnish
```

```
X-Powered-By: Express
X-Varnish: 305927378
Content-Length: 339
Connection: keep-alive
```

Berikut detail *response body* permintaan detail event yang dibalas oleh Web API:

```
{
  "url": "http://localhost:3000/event1",
  "data": {
    "_id": 1,
    "title": "Free event",
    "shortDescription": "Free event for students",
    "times": [
      {
        "milliseconds": 1613028781634,
        "fullFormat": "Thu Feb 11 2021 14:33:01\nGMT+0700 (WIT)",
        "year": 2021,
        "month": 1,
        "date": 11,
        "day": 1
      }
    ],
    "isPassed": false,
    "rating": 3,
    "lastUpdated": 1612928781634,
    "timestamp": 1612928781634
  }
}
```

### C.1.2 Hasil Respon Permintaan Detail Event dalam Format XML

Berikut detail *request header* ketika melakukan permintaan detail event dalam format XML:

```
Accept: application/xml
User-Agent: Mozilla/5.0 (X11; Linux i686)
```

```
AppleWebKit/537.31 (KHTML, like Gecko)
Chrome/26.0.1410.63 Safari/537.31
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3
```

Berikut detail *response header* permintaan detail event yang dibalas oleh Web API:

```
Accept-Ranges: bytes
Age: 0
Content-Type: application/xml
Date: Mon, 22 Apr 2013 19:01:40 GMT
Server: nginx
Vary: Accept
Via: 1.1 varnish
X-Powered-By: Express
X-Varnish: 305927378
Content-Length: 339
Connection: keep-alive
```

Berikut detail *response body* permintaan detail event yang dibalas oleh Web API:

```
<response>
  <url>http://localhost:3000/event1</url>
  <data id="1">
    <title>Free event</title>
    <shortDescription>Free event for students
      </shortDescription>
    <times>
      <time>
        <milliseconds>1613028781634</milliseconds>
        <fullFormat>Thu Feb 11 2021 14:33:01
          GMT+0700 (WIT)</fullFormat>
        <year>2021</year>
        <month>1</month>
        <date>11</date>
        <day>1</day>
      </time>
```

```
</times>
<isPassed>false </isPassed>
<rating>3</rating>
<lastUpdated>1612928781634</lastUpdated>
<timestamp>1612928781634</timestamp>
</data>
</response>
```

## C.2 Hasil Respon Permintaan Mengambil Daftar Event

### C.2.1 Hasil Respon Permintaan Daftar Event dalam Format JSON

Berikut detail *request header* ketika melakukan permintaan daftar event dalam format JSON:

```
Accept: application/json
User-Agent: Mozilla/5.0 (X11; Linux i686)
  AppleWebKit/537.31 (KHTML, like Gecko)
  Chrome/26.0.1410.63 Safari/537.31
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3
```

Berikut detail *response header* permintaan daftar event yang dibalas oleh Web API:

```
Accept-Ranges: bytes
Age: 0
Content-Type: application/json
Date: Mon, 22 Apr 2013 19:01:40 GMT
Server: nginx
Vary: Accept
Via: 1.1 varnish
X-Powered-By: Express
X-Varnish: 305927378
Content-Length: 339
Connection: keep-alive
```

Berikut detail *response body* permintaan daftar event yang dibalas oleh Web API:

```
{
  "url": "http://localhost:3000/events",
  "data": [
    {
      "_id": 1,
      "title": "Free event",
      "shortDescription": "Free event for students",
      "times": [
        {
          "milliseconds": 1613028781634,
          "fullFormat": "Thu Feb 11 2021 \n\
14:33:01 GMT+0700 (WIT)",
          "year": 2021,
          "month": 1,
          "date": 11,
          "day": 1
        }
      ],
      "isPassed": false,
      "rating": 3,
      "lastUpdated": 1612928781634,
      "timestamp": 1612928781634
    },
    {
      "_id": 2,
      "title": "Dolor zzril exerci.",
      "shortDescription": "Lobortis \n\
duis accumsan at vulputate.",
      "times": [
        {
          "milliseconds": 1613028781634,
          "fullFormat": "Thu Feb 11 2021\n\
14:33:01 GMT+0700 (WIT)",
          "year": 2021,
          "month": 1,
```



```

        "date": 11,
        "day": 1
    }
],
"isPassed": true ,
"rating": 4,
"lastUpdated": 161292898164,
"timestamp": 1612928781634
},
{
    "_id": 3,
    "title": "Accumsan hendrerit vel.",
    "shortDescription": "Vulputate\n\
        suscipit nibh nulla qui.",
    "times": [
        {
            "milliseconds": 1613028781634,
            "fullFormat": "Thu Feb 11 2021\n\
                14:33:01 GMT+0700 (WIT)",
            "year": 2021,
            "month": 1,
            "date": 11,
            "day": 1
        }
    ],
    "isPassed": false ,
    "rating": 3,
    "lastUpdated": 161292898164,
    "timestamp": 1612928781634
},
// dan seterusnya sampai item ke-15
]
}

```

## C.2.2 Hasil Respon Permintaan Daftar Event dalam Format XML

Berikut detail *request header* ketika melakukan permintaan daftar event dalam format JSON:

```
Accept: application/json
User-Agent: Mozilla/5.0 (X11; Linux i686)
  AppleWebKit/537.31 (KHTML, like Gecko)
  Chrome/26.0.1410.63 Safari/537.31
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3
```

Berikut detail *response header* permintaan daftar event yang dibalas oleh Web API:

```
Accept-Ranges: bytes
Age: 0
Content-Type: application/json
Date: Mon, 22 Apr 2013 19:01:40 GMT
Server: nginx
Vary: Accept
Via: 1.1 varnish
X-Powered-By: Express
X-Varnish: 305927378
Content-Length: 339
Connection: keep-alive
```

Berikut detail *response body* permintaan daftar event yang dibalas oleh Web API:

```
<response>
  <url>http://localhost:3000/events</url>
  <data>
    <data id="1">
      <title>Free event</title>
      <shortDescription>Free
        event for students</shortDescription>
      <times>
        <time>
          <milliseconds>1613028781634</milliseconds>
```

```

        <fullFormat>Thu Feb 11 2021
          14:33:01 GMT+0700 (WIT)</fullFormat>
        <year>2021</year>
        <month>1</month>
        <date>11</date>
        <day>1</day>
      </time>
    </times>
    <isPassed>false </isPassed>
    <rating>3</rating>
    <lastUpdated>1612928781634</lastUpdated>
    <timestamp>1612928781634</timestamp>
  </data>
  <data id="2">
    <title>Dolor zzril exerci.</title>
    <shortDescription>Lobortis
      dui accumsan at vulputate.</shortDescription>
    <times>
      <time>
        <milliseconds>1613028781634</milliseconds>
        <fullFormat>Thu Feb 11 2021 14:33:01
          GMT+0700 (WIT)</fullFormat>
        <year>2021</year>
        <month>1</month>
        <date>11</date>
        <day>1</day>
      </time>
    </times>
    <isPassed>true </isPassed>
    <rating>4</rating>
    <lastUpdated>161292898164</lastUpdated>
    <timestamp>1612928781634</timestamp>
  </data>
  <data id="3">
    <title>Accumsan hendrerit vel.</title>
    <shortDescription>Vulputate
      suscipit nibh nulla qui.</shortDescription>

```

```
<times>
  <time>
    <milliseconds>1613028781634</milliseconds>
    <fullFormat>Thu Feb 11 2021 14:33:01
      GMT+0700 (WIT)</fullFormat>
    <year>2021</year>
    <month>1</month>
    <date>11</date>
    <day>1</day>
  </time>
</times>
<isPassed>false </isPassed>
<rating>3</rating>
<lastUpdated>161292898164</lastUpdated>
<timestamp>1612928781634</timestamp>
</data>
</data>
</response>
```

## Lampiran D

### Hasil Pengujian Web API

Berikut detail keluaran log hasil pengujian di level *acceptance* yang dijalankan dengan perintah *make test*<sup>1</sup>:

```
npm install
npm install -d
npm info it worked if it ends with ok
npm info using npm@1.2.21
npm info using node@v0.10.7
npm info shrinkwrap
  file "/home/ghazali/version-control/
  git/kuliah/skripsi/ghanoz-json/npm-shrinkwrap.json"
npm info preinstall ghanoz-json@0.3.5
npm info shrinkwrap
  file "/home/ghazali/version-control/
  git/kuliah/skripsi/ghanoz-json/npm-shrinkwrap.json"
npm info build
  /home/ghazali/version-control/
  git/kuliah/skripsi/ghanoz-json
npm info linkStuff ghanoz-json@0.3.5
npm info install ghanoz-json@0.3.5
npm info postinstall ghanoz-json@0.3.5
npm info prepublish ghanoz-json@0.3.5
npm info ok
mongo ghanozjson_test --eval "db.dropDatabase()"
MongoDB shell version: 2.4.3
connecting to: ghanozjson_test
```

<sup>1</sup><https://github.com/muhammadghazali/ghanoz-json/blob/master/Makefile>

```

[object Object]
mongoimport --jsonArray
  --host localhost
  --db ghanozjson_test
  --collection events --file test/db/events.json
connected to: localhost
Thu May 23 09:17:33.533 check 9 150
Thu May 23 09:17:33.534 imported 150 objects
NODE_ENV=testing npm test

> ghanoz-json@0.3.5 test
  /home/ghazali/version-control/
  git/kuliah/skripsi/ghanoz-json
> node_modules/vows/bin/vows
  test/api-end-point-test/*-spec.js

Express server listening on port 3000
127.0.0.1 - - [Thu, 23 May 2013 02:17:34 GMT]
  "GET /event/1 HTTP/1.1" 200 356 "-" "-"
    127.0.0.1 - - [Thu, 23 May 2013 02:17:34 GMT]
  "GET /event/1 HTTP/1.1" 200 713 "-" "-"
127.0.0.1 - - [Thu, 23 May 2013 02:17:34 GMT]
  "GET /event/1 HTTP/1.1" 200 713 "-" "-"
    127.0.0.1 - - [Thu, 23 May 2013 02:17:34 GMT]
  "GET /event/1 HTTP/1.1" 406 49 "-" "-"
    127.0.0.1 - - [Thu, 23 May 2013 02:17:34 GMT]
  "GET /events HTTP/1.1" 200 4722 "-" "-"
    127.0.0.1 - - [Thu, 23 May 2013 02:17:34 GMT]
  "GET /events HTTP/1.1" 200 10870 "-" "-"
127.0.0.1 - - [Thu, 23 May 2013 02:17:34 GMT]
  "GET /events HTTP/1.1" 200 10870 "-" "-"
    127.0.0.1 - - [Thu, 23 May 2013 02:17:34 GMT]
  "GET /events HTTP/1.1" 406 49 "-" "-"

    OK      10 honored (0.126s)
  mongodump --db ghanozjson_test
connected to: 127.0.0.1

```

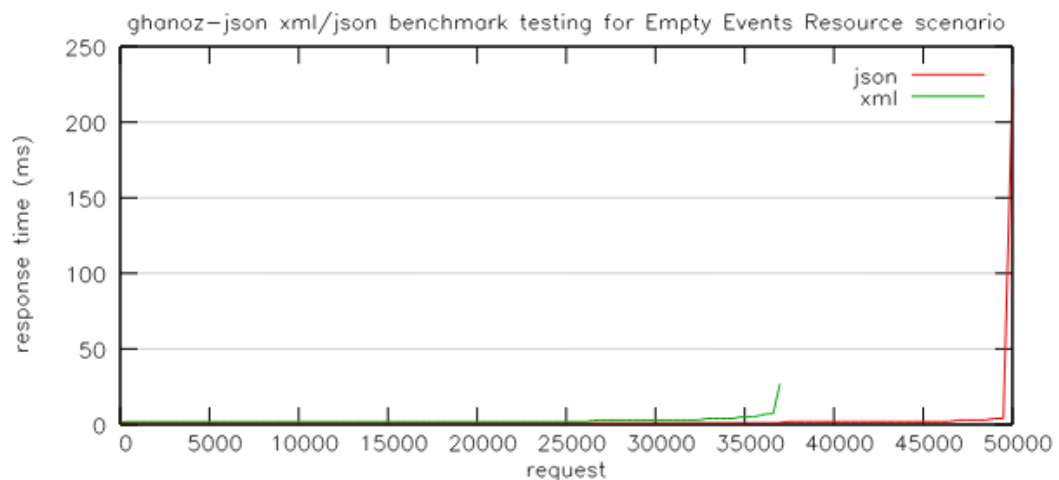
```
Thu May 23 09:17:34.254 DATABASE:
  ghanozjson_test to dump/ghanozjson_test
Thu May 23 09:17:34.255
  ghanozjson_test.system.indexes
  to dump/ghanozjson_test/system.indexes.bson
Thu May 23 09:17:34.256 1 objects
Thu May 23 09:17:34.256
  ghanozjson_test.events
  to dump/ghanozjson_test/events.bson
Thu May 23 09:17:34.258 150 objects
Thu May 23 09:17:34.258
  Metadata for ghanozjson_test.events
  to dump/ghanozjson_test/events.metadata.json
mongo ghanozjson_test --eval "db.dropDatabase()"
MongoDB shell version: 2.4.3
connecting to: ghanozjson_test
[object Object]
```

# Lampiran E

## Hasil Perbandingan Ukuran Data

### E.1 Empty Resource

*Empty Resource.* Melakukan perbandingan jumlah permintaan yang berhasil dire-  
spon dengan respon yang dikembalikan berisi ”resource kosong” dengan setiap *field*  
dari *resource* diberi *string* kosong. Representasi dalam bentuk grafik<sup>1</sup> dari hasil  
perbandingan tersebut dapat dilihat pada gambar E.1.



Gambar E.1: Perbandingan Jumlah Permintaan yang berhasil direspon untuk skenario Empty Resource

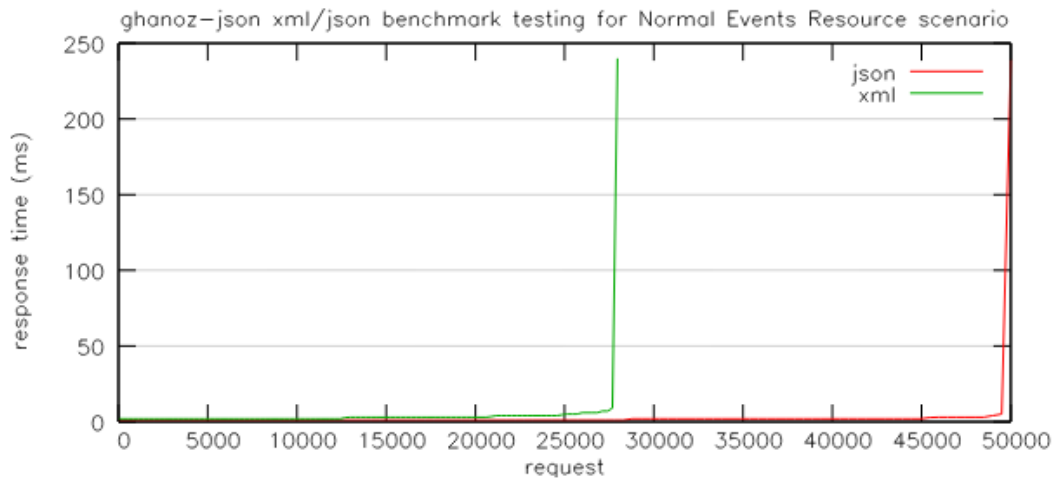
Bisa dilihat melalui grafik di atas, bahwa dalam 73,853 detik, Web API mampu mengembalikan respon dari 50000 permintaan dalam format JSON. Sedangkan dalam 90 detik, Web API hanya mampu mengembalikan respon dari 36973 permintaan dalam format XML.

<sup>1</sup>Grafik dibuat dengan menggunakan kakas gnuplot <http://www.gnuplot.info/>



## E.2 Normal Resource

*Normal Resource.* Melakukan perbandingan jumlah permintaan yang berhasil direspon dengan respon yang dikembalikan berisi *resource* dengan setiap *field* dari *resource* diberi data yang "normal". Representasi dalam bentuk grafik<sup>2</sup> dari hasil perbandingan tersebut dapat dilihat pada gambar E.2.



Gambar E.2: Perbandingan Jumlah Permintaan yang berhasil direspon untuk skenario Normal Resource

Bisa dilihat melalui grafik di atas, bahwa dalam 87.414 detik, Web API mampu mengembalikan respon dari 50000 permintaan dalam format JSON. Sedangkan dalam 90 detik, Web API hanya mampu mengembalikan respon dari 27966 permintaan dalam format XML.

## E.3 Just Once

*Just Once.* Melakukan perbandingan ukuran data terhadap hasil respon yang dikembalikan berisi *resource* dengan setiap *field* dari *resource* diberi data yang "normal". Berikut detail hasil pembandingannya:

1. Ukuran data *resource* dalam format JSON dengan besar 4377 *bytes* berhasil direspon oleh Web API dalam waktu 0,0004 detik
2. Ukuran data *resource* dalam format XML dengan besar 6635 *bytes* berhasil direspon oleh Web API dalam waktu 0,0005 detik

<sup>2</sup>Grafik dibuat dengan menggunakan kakas gnuplot <http://www.gnuplot.info/>