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>

15 Mei 2013

Daftar Isi

Daftar Gambar

Daftar Tabel

Ringkasan

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[1] 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*[2] yang tersimpan di salah satu layanan *Database as a Service*[3] yang digunakan oleh LLM di AppFog[4]. 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 pengelohan tersebut dinamakan serialisasi data[5].

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

[1] http://en.wikipedia.org/wiki/Web_API
[2] http://en.wikipedia.org/wiki/Data_store
[3] http://en.wikipedia.org/wiki/Cloud_database
[4] http://www.appfog.com/
[5] Lihat bagian Landasan teori: Serialiasi Data

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[1] 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*[2] yang tersimpan di salah satu layanan *Database as a Service*[3] yang digunakan oleh LLM di AppFog[4]. 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[5].

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,

[1] http://en.wikipedia.org/wiki/Web_API
[2] http://en.wikipedia.org/wiki/Data_store
[3] http://en.wikipedia.org/wiki/Cloud_database
[4] http://www.appfog.com/
[5] 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[6] 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 pengembangkan perangkat lunak *Agile*.
4. Kesimpulan. Pada tahap ini penulis menganalisa hasil pengujian hipotesis.

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

1. Untuk memungkinkan pengguna Anda mengakses data mereka melalui aplikasi pihak ketiga. Pertimbangkan berapa banyak pihak ketiga Twitter klien yang ada[1], semua menggunakan API Twitter[2], daripada mengakses langsung

[1] http://en.wikipedia.org/wiki/List_of_Twitter_services_and_applications
[2] https://dev.twitter.com/

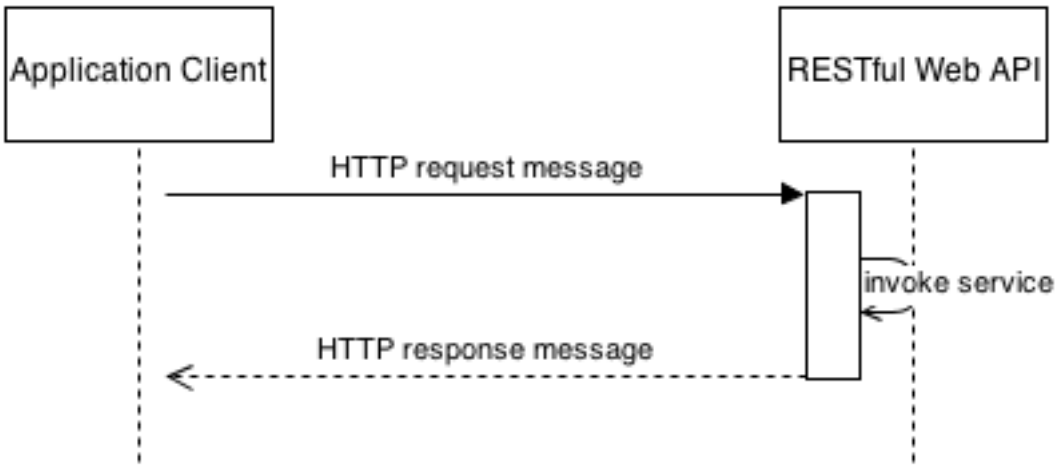
ke situs *web*[3]. Hal yang sama berlaku untuk Amazon[4] dan eBay[5], antara lain, yang memungkinkan pengguna untuk mengakses data katalog mereka dan bahkan mengeksekusi penjualan, semua melalui API.

2. Untuk memungkinkan pengembang aplikasi mobile untuk mengakses situs Anda dengan mengirim dan mengambil data menggunakan HTTP.

3. Untuk memungkinkan aplikasi Anda sendiri untuk mengakses data sendiri melalui panggilan Ajax[6]. Bila Anda membuat panggilan JavaScript di latar belakang menggunakan Ajax, kemungkinan besar Anda akan ingin menggunakan panggilan API, 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[7]. *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:



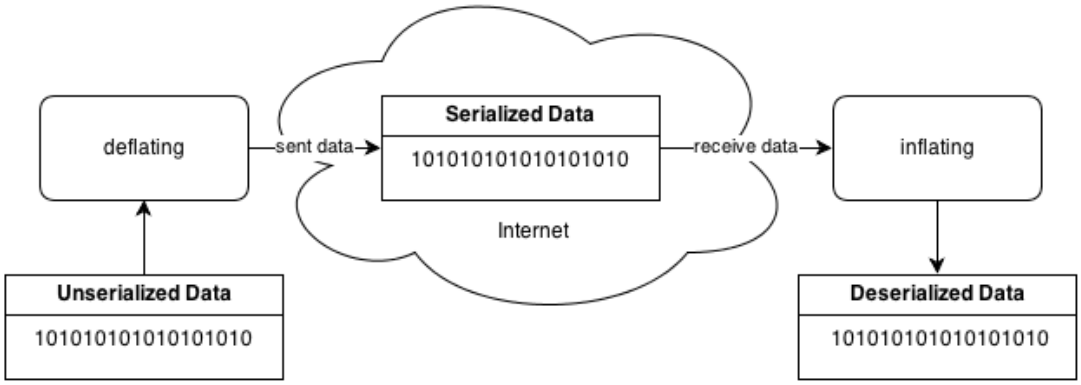
Gambar 2.1: RESTful Web API

[3]https://twitter.com/
[4]http://www.amazon.com/
[5]http://www.ebay.com/
[6]https://developer.mozilla.org/en/docs/AJAX
[7]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[8].

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[9]:

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

[8]Web Services Directory http://www.programmableweb.com/apis/directory/1?protocol=REST
[9]http://en.wikipedia.org/wiki/Comparison_of_data_serialization_formats

5. BSON
6. *D-Bus Message Protocol*
7. JSON
8. MessagePack
9. Netstrings
10. OGDL
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*[10]. Relatif sangat mudah bagi manusia untuk membaca dan menulis JSON. Relatif sangat mudah pula bagi mesin (komputer) untuk mengurai dan menghasilkan JSON.

[10]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[11]. 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*.

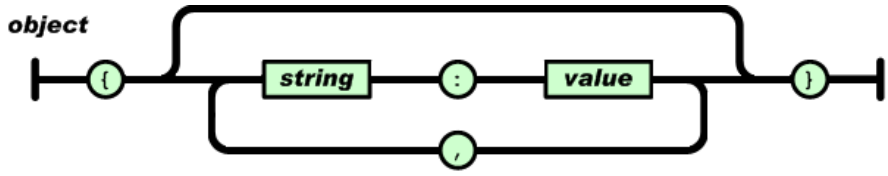
2. *Ordered List*. Dalam kebanyakan bahasa, struktur ini direalisasikan sebagai *array*, *vector*, *list*, atau *sequence*.

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 primtif (*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*.



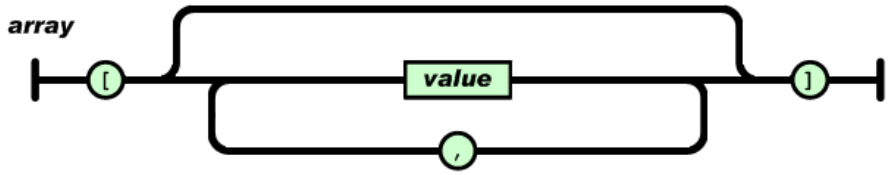
Gambar 2.3: Diagram Sintaks untuk Tipe Object [6]

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

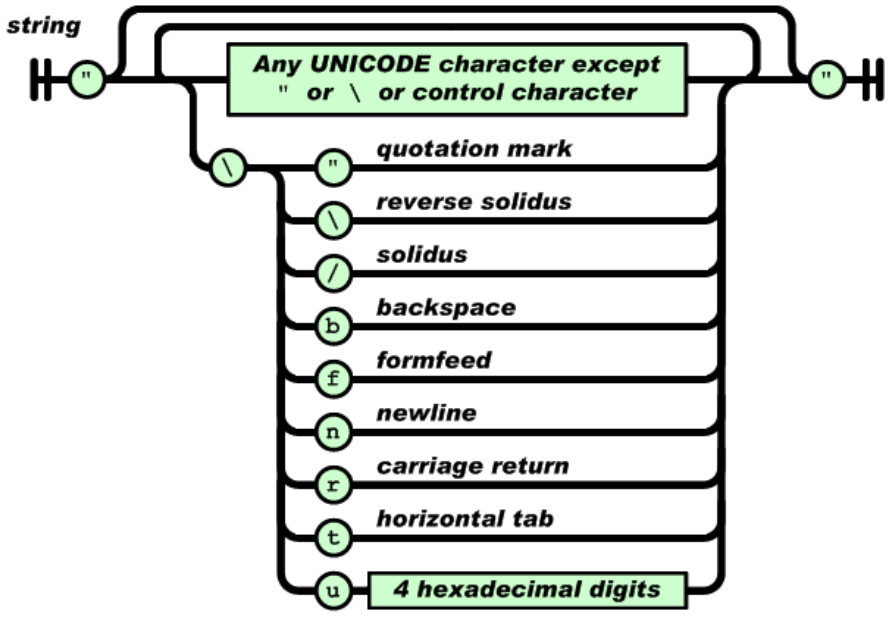
Value dapat berupa *string* dalam tanda kutip ganda, *number*, *boolean*, *object* atau *array*. Struktur ini dapat diulang.

String adalah rangkaian yang terdiri dari nol atau lebih karakter *Unicode*[12].

[11]http://www.ecma-international.org/publications/standards/Ecma-262-arch.htm
[12]http://www.unicode.org/versions/Unicode6.2.0/

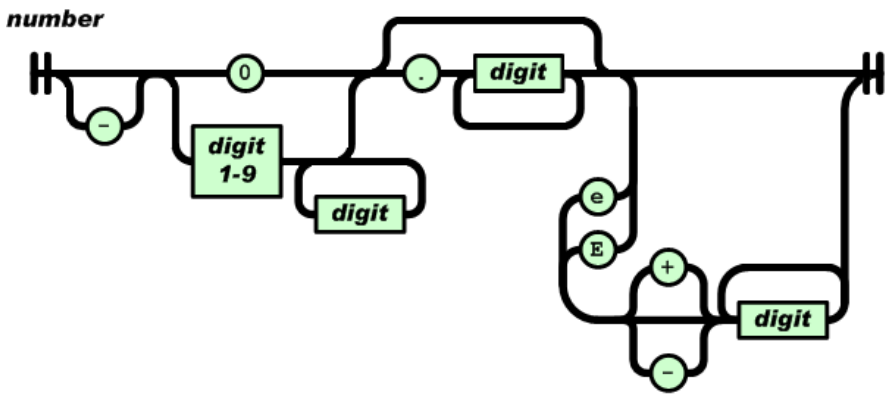


Gambar 2.4: Diagram Sintaks untuk Tipe Array [6]



Gambar 2.5: Diagram Sintaks untuk Tipe String [6]

Number di JSON sama seperti tipe data *number* di C atau Java, kecuali format oktal[13] dan heksadesimal[14] tidak digunakan.



Gambar 2.6: Diagram Sintaks untuk Tipe Number [6]

[13]Contoh angka dalam bentuk oktal 061 062 063
[14]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[15] 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[16]:

```
<?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>
```

[15] http://www.w3.org/TR/REC-xml/
[16] 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 kelebihannya 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*[17], di mana persyaratan dan solusi berkembang melalui kolaborasi. Ini mempromosikan perencanaan adaptif, perkembangan dan pengiriman yang evolusioner, pendekatan *time-boxed*[18] 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[19] 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*[20] dan *pair programming*[21].

[17] http://en.wikipedia.org/wiki/Iterative_and_incremental_development
[18] http://en.wikipedia.org/wiki/Timeboxing
[19] 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
[20] http://en.wikipedia.org/wiki/Colocation_(business)
[21] 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 yang dapat mengorganisir diri sendiri.

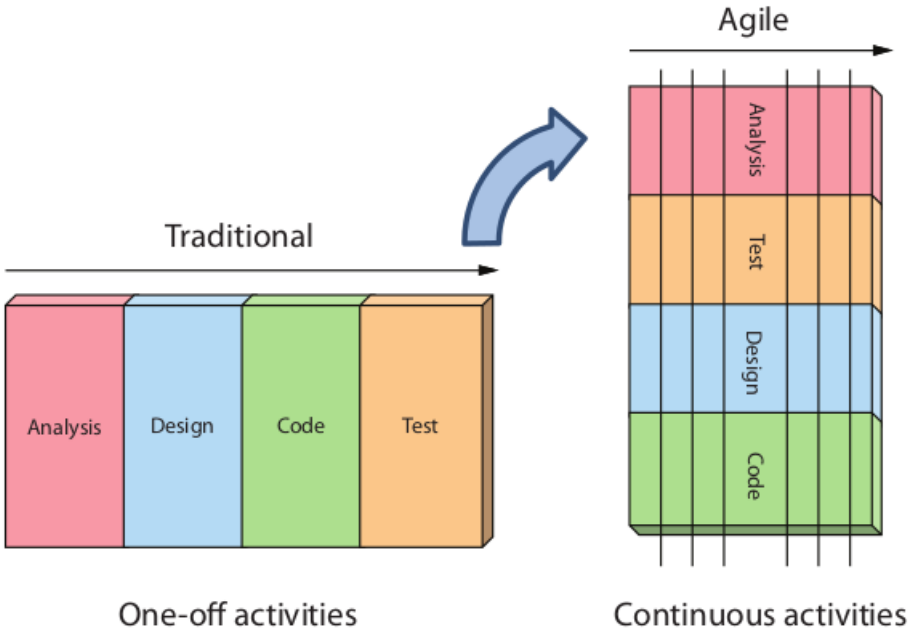
12. Secara berkala, tim pengembang berefleksi tentang bagaimana untuk menjadi lebih efektif, kemudian menyesuaikan dan menyelaraskan kebiasaan bekerja mereka.

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

Bab 3

Analisis Sistem

3.1 Roadmap Pengerjaan Purwa-rupa Web API

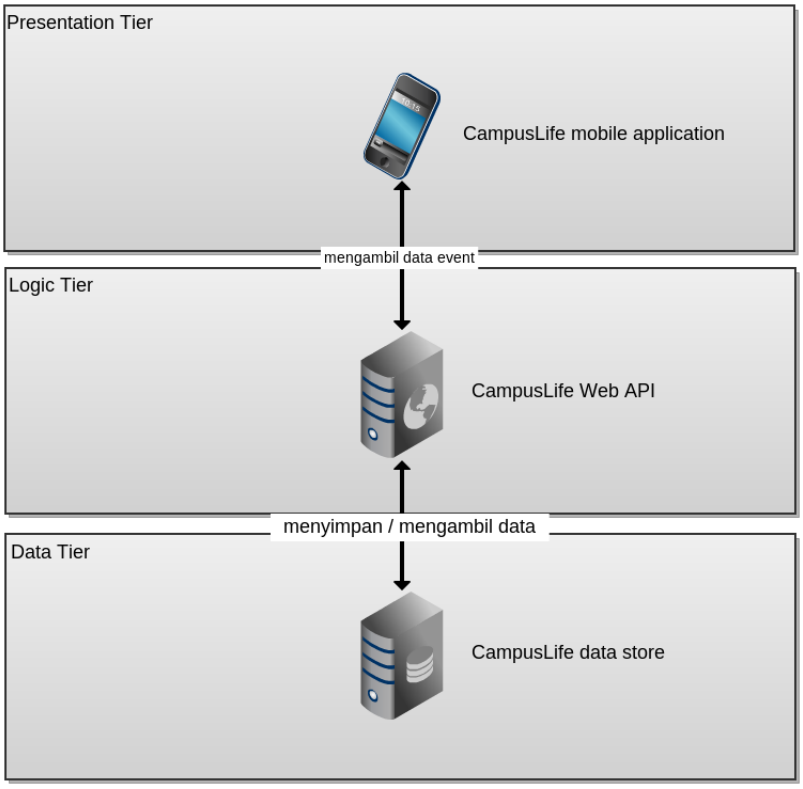
Sebelum penulis memulai *software development life cycle*, penulis terlebih dahulu menentukkan *roadmap* dari pengembangan Web API yang terdiri dari 3 *milestone* yang harus dipenuhi. Berikut daftar *milestone* tersebut:

1. M 1: *Application Skeleton*. Pada *milestone* ini penulis mempersiapkan struktur aplikasi[1]
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

[1] 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[2] 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

[2]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*[3] 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

[3]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.

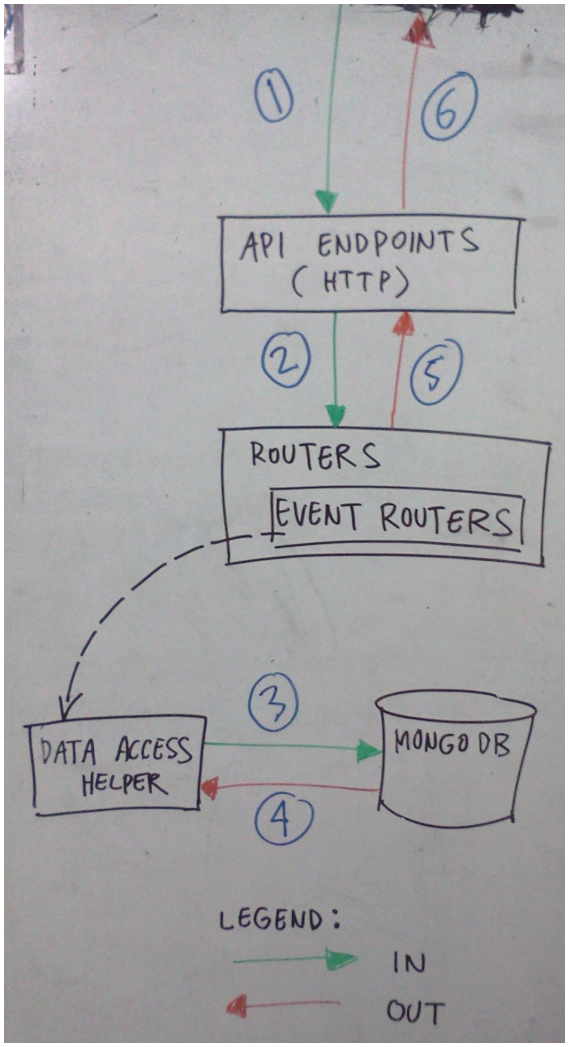
Bab 4

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 Anda menemukan model yang diperlihatkan di bab ini mungkin tidak ”benar” menurut perspektif Anda 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

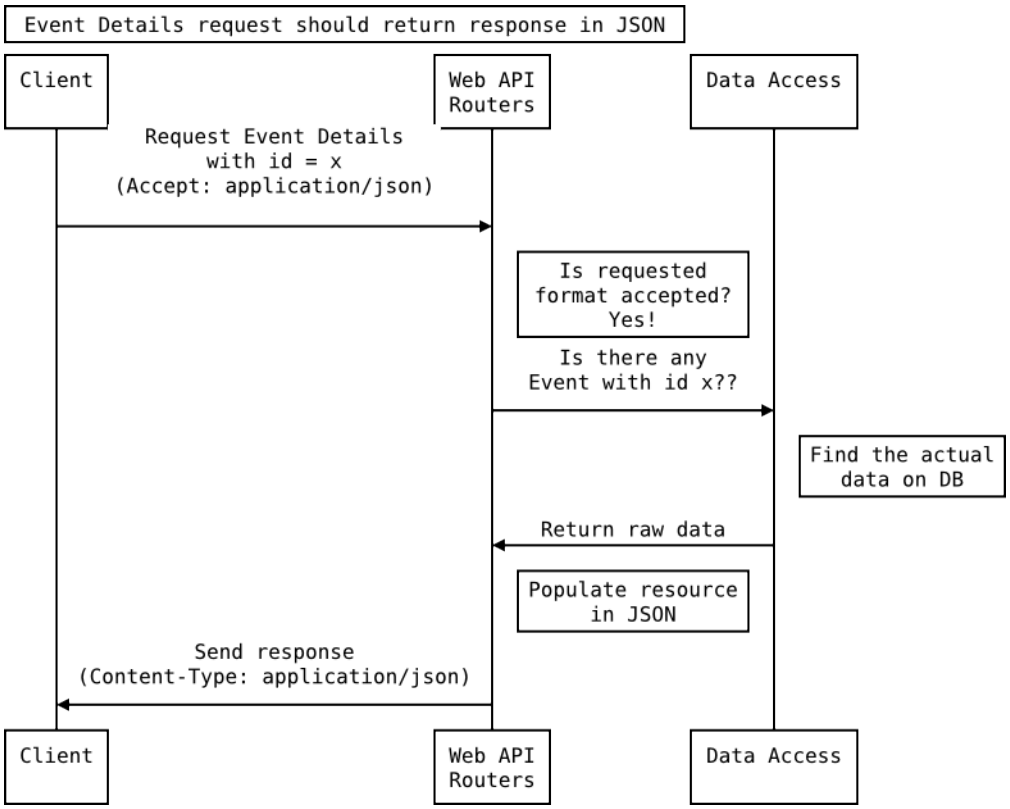
Pada bagian ini akan berisi laporan mengenai hasil pemodelan yang penulis lakukan saat melakukan perancangan sistem. Untuk mendapatkan gambaran dengan jelas mengenai sistem yang akan dibangun, penulis melakukan pemodelan interaksi antara *client* dan Web API *server*. Model tersebut dibuat dalam bentuk *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*[1].

[1] http://www.agilemodeling.com/artifacts/sequenceDiagram.htm

4.2.1 Permintaan Detail Event dalam format JSON

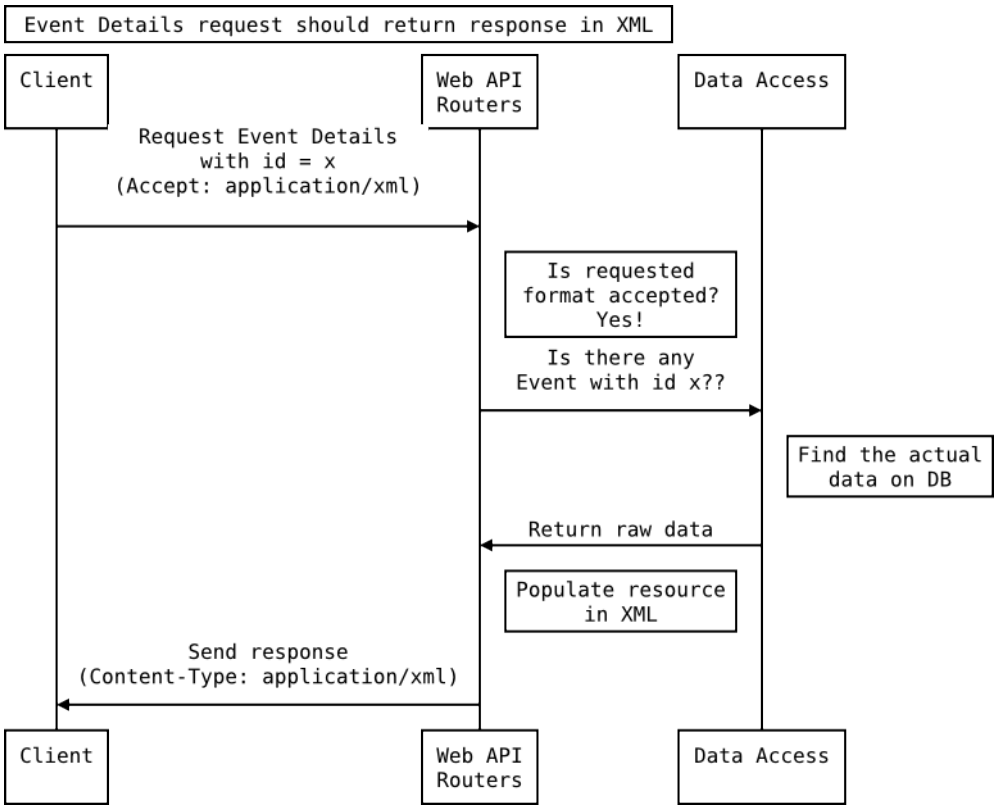
Pemodelan interaksi antara *client* dan Web API *server* saat *client* meminta detail event dalam format JSON dapat dilihat pada gambar 4.2.



Gambar 4.2: Permintaan Detail Event dalam Format JSON

4.2.2 Permintaan Detail Event dalam format XML

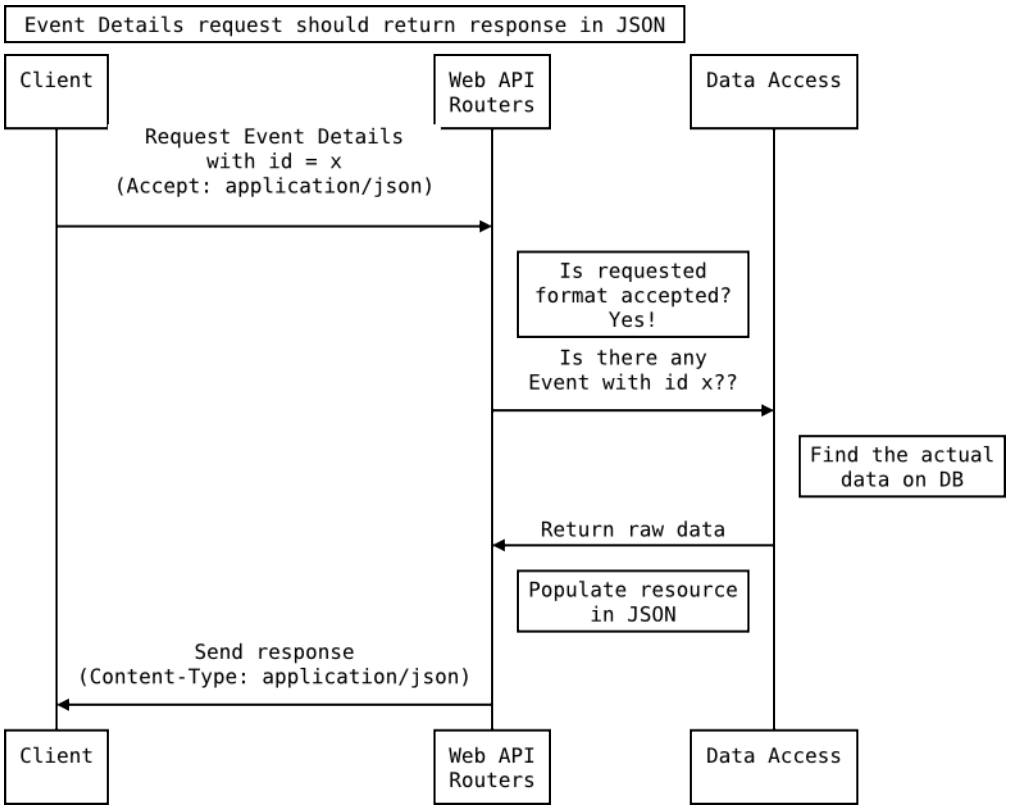
Pemodelan interaksi antara *client* dan Web API *server* saat *client* meminta detail event dalam format XML dapat dilihat pada gambar 4.3.



Gambar 4.3: Permintaan Detail Event dalam Format XML

4.2.3 Permintaan Daftar Event dalam format JSON

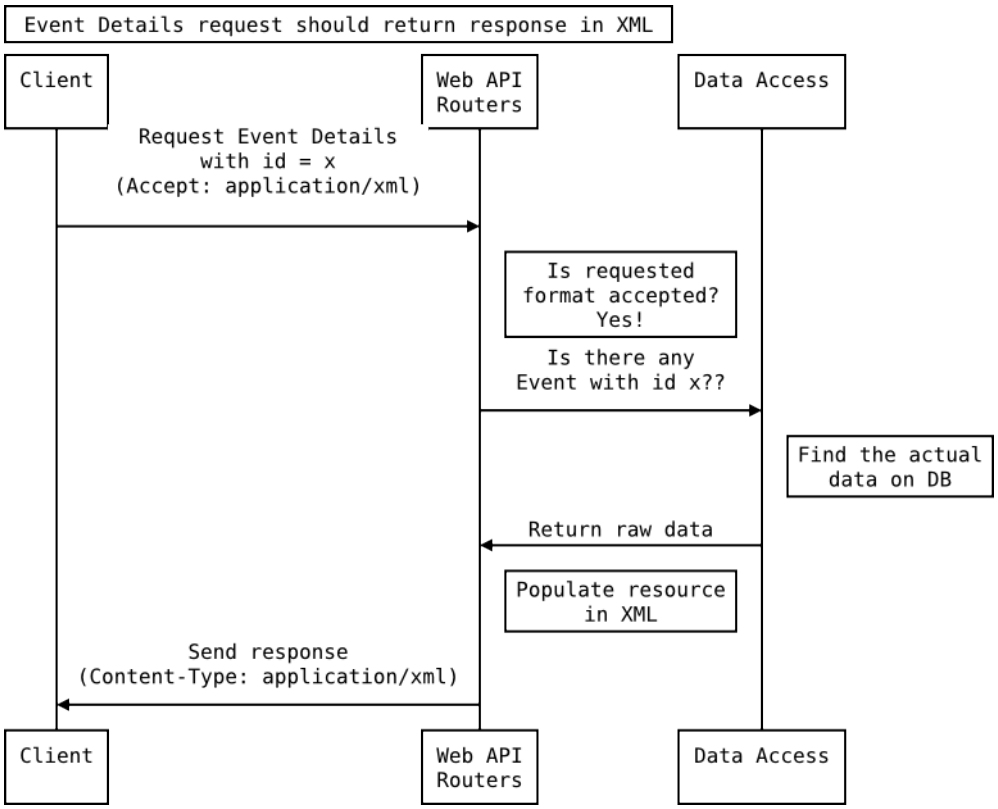
Pemodelan interaksi antara *client* dan Web API *server* saat *client* meminta daftar event dalam format JSON dapat dilihat pada gambar 4.4.



Gambar 4.4: Permintaan Daftar Event dalam Format JSON

4.2.4 Permintaan Daftar Event dalam format XML

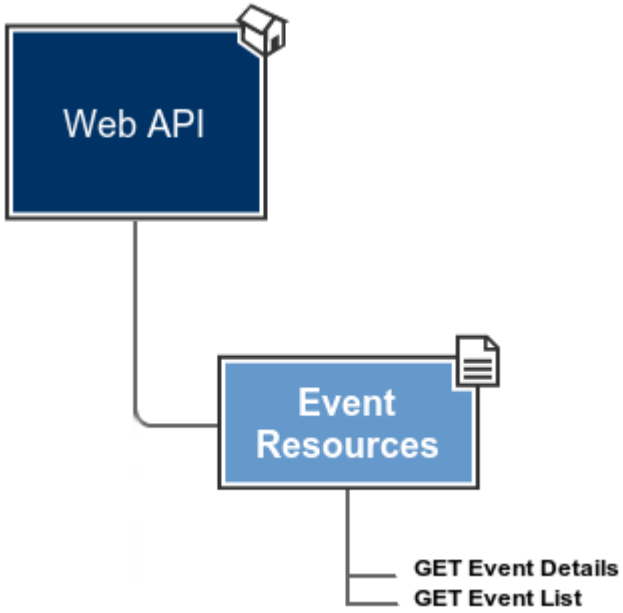
Pemodelan interaksi antara *client* dan Web API *server* saat *client* meminta daftar event dalam format XML dapat dilihat pada gambar 4.5.



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.

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 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 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_inte diakses pada 22 Maret 2013.

[10] *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 diakses pada 7 Mei 2013.

[20] *Manifesto Pengembangan Perangkat Lunak Agile* http://agilemanifesto.org/iso/id/ diakses pada 8 April 2013.

[21] *The Agile Samuari* 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 reponse 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*[1]

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>
```

[1]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*[2]

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,
```

[2]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*[3]

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>
```

[3]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[4]

[4]https://gist.github.com/muhammadghazali/5409528#file-event-list-xml