RESTful Web Service dengan HTTP dan JSON

Pascal Alfadian Nugroho, S.Kom, M.Comp

Daftar Isi

Prakata

Buku ini berguna bagi Anda yang ingin membuat aplikasi *client-server* modern. Walaupun terdapat banyak cara berkomunikasi antara *client* dengan *server*, *web service* menawarkan cara komunikasi yang sederhana, kompatibel dengan banyak *platform*, serta dapat digunakan dalam berbagai jenis jaringan (sangat penting di era *mobile* saat ini).

Bab 1 melihat kembali protokol HTTP yang biasa digunakan dalam menyajikan halaman web. Bab 2 mengajarkan bagaimana memanfaatkan protokol tersebut untuk komunikasi aplikasi selain web. Bab 3 membahas bagaiaman menggunakan protokol tersebut untuk komunikasi secara rapi dengan arsitektur REST. Bab 4 mengenalkan beberapa *web service* gratis yang tersedia di internet (yang tentu saja memanfaatkan protokol HTTP). Bab 5 memperkenalkan implementasi *web service* di beberapa jenis platform.

Penulis berharap, setelah membaca buku ini pembaca dapat membuat aplikasi sendiri yang bersifat *client-server* serta *multi-platform*, seperti layaknya aplikasi-aplikasi masa kini.

Selamat belajar!

Kata Sambutan

Di era teknologi informasi modern, penggunaan jaringan komputer menjadi suatu kebutuhan yang tak terelakkan, antara lain dalam aplikasi bisnis. Pengetahuan tentang komunikasi antara client dan server, dan bagaimana membangun aplikasi client-server yang baik, dalam arti aman, sederhana, andal tetapi murah menjadi suatu kebutuhan. Pengetahuan tentang pembuatan apliaksi client-server menjadi suatu materi pokok mahasiswa yang wajib dikuasai oleh mahasiswa Program Studi Teknik Informatika. Oleh karena itu penerbitan buku “RESTful Web Service dengan HTTP dan JSON” perlu disambut dengan antusias.

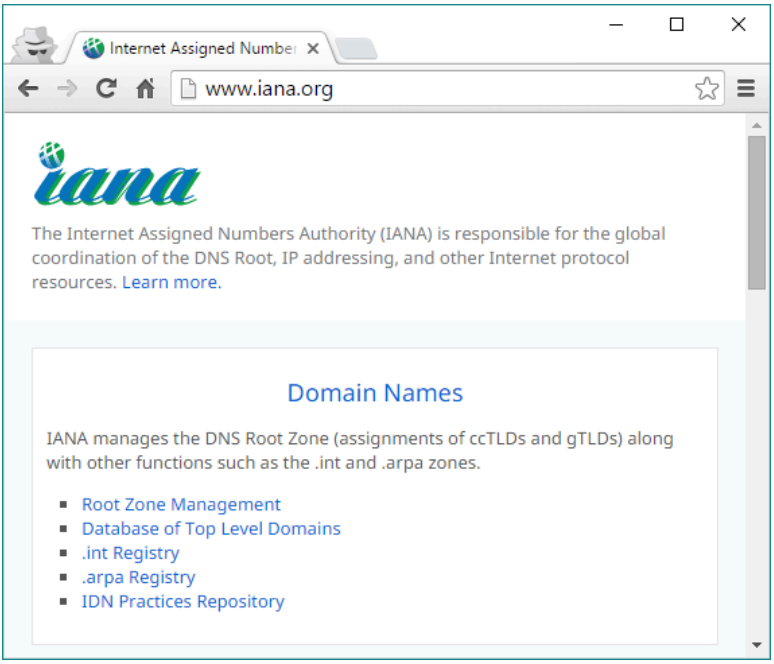
Buku ini sangat perlu dibaca oleh dosen, mahasiswa dan praktisi yang memerlukan pegangan dalam mengajar, belajar ataupun membuat aplikasi client-server modern karena ditulis oleh seorang akademisi dan praktisi yang berpengalaman dan berkompeten dibidangnya. Atas nama Fakultas Teknologi Informasi dan Sains, Universitas Katolik Parahyangan, saya memberikan apresiasi yang sangat tinggi kepada Penulis, Pascal Alfadian Nogroho, rekan kerja di Fakultas Teknologi Informasi dan Sains, Universitas Katolik Parahyangan atas sumbangan pemikiran dan pengalamannya, baik sebagai pengajar maupun sebagai praktisi dalam pembuatan aplikasi client-server modern. Semoga buku ini bermanfaat, dapat memperkaya pengetahuan para pembacanya, dan khsusnya dapat menjadi pegangan dalam mengajar dan mempelajari pembuatan aplikasi client-server modern.

Bandung, 12 Mei 2017
Dr. Ferry Jaya Permana
Dekan Fakultas Teknologi Informasi dan Sains,
Universitas Katolik Parahyangan

1

Mengenal Kembali HTTP

Anda pasti sudah biasa mengakses situs web, katakanlah mengakses situs web pada alamat http://www.iana.org/. Saat Anda membukanya di aplikasi *browser* (peramban), Anda akan menemui halaman seperti gambar di bawah ini.



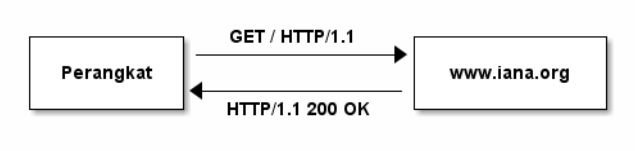
Gambar 1.1. Halaman situs http://www.iana.org/

Di belakang kesederhanaan halaman yang ditampilkan, sebenarnya terjadi berbagai macam pertukaran data antara perangkat Anda dengan *server* web www.iana.org. Pertukaran data ini menggunakan protokol HTTP.

1.1. Arsitektur Protokol HTTP

Protokol HTTP merupakan protokol yang berjalan di atas protokol TCP pada *port* 80. Saat Anda membuka halaman http://www.iana.org/, yang terjadi adalah sebagai berikut:

1. Perangkat melakukan koneksi di *port* 80 ke *server* di alamat *www.iana.org*
2. Perangkat meminta konten dari halaman utama (pada *path /*)
3. *Server* mengirimkan balik konten dalam format HTML
4. Koneksi pada *port* 80 selesai (ditutup)
5. *Browser* mencari pada konten HTML yang didapat. Jika ada elemen lain yang harus diunduh (gambar, video, dll). Untuk setiap elemen, langkah 1-4 kembali dilakukan.



Gambar 1.2. Komunikasi Perangkat dengan iana

Pada kasus di atas perangkat berperan sebagai *client*, atau dalam standar RFC 7230 dikenal sebagai *User Agent*. Dalam standar yang sama, *server* iana.org dikenal sebagai *Origin Server*. Dalam buku ini, kita akan menggunakan istilah *client* dan *server* untuk kedua entitas tersebut.

Protokol HTTP juga mendukung koneksi persisten, di mana koneksi tidak ditutup setelah konten dikirimkan. Dalam buku ini, kita asumsikan koneksi akan selalu ditutup.

1.2. Format Permintaan

Pada skenario di atas, *client* mengirimkan data berupa teks kepada *server* (*request*) seperti berikut:

```
GET / HTTP/1.1 ❶
Host: www.iana.org ❷
Connection: keep-alive
```

```
Cache-Control: max-age=0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/
webp,*/*;q=0.8
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36
 (KHTML, like Gecko) Chrome/48.0.2564.109 Safari/537.36
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8,id;q=0.6
If-Modified-Since: Fri, 18 Dec 2015 22:42:10 GMT
❸
```

❶ Baris pertama pada *request* disebut dengan *request line* dan diawali dengan *request method* (metode permintaan), dalam kasus ini, `GET`. *Request method* diikuti dengan *resource* (sumberdaya) yang diinginkan, dalam kasus ini `/`. *Request line* diakhiri dengan versi protokol yang digunakan, dalam kasus ini `HTTP/1.1`.

❷ Baris kedua dan baris-baris berikutnya, sampai ditemukan baris kosong, berisi *request headers* dalam format "*nama-header: nilai-header*". Pada contoh di atas, terdapat header `Host` berisi *www.iana.org* yang menandakan bahwa *browser* ingin mengakses situs yang dikenal dari alamat *www.iana.org*.

❸ Perhatikan bahwa ada ekstra satu baris kosong di akhir *request*. Pada *request*, baris kosong memisahkan antara *request headers* dengan *request body* (tubuh permintaan). Pada kasus di atas, *request* tidak mengandung *body*. Oleh karena itu, baris kosong di atas menandakan akhir dari *request*.

1.2.1. Request Method (Metode Permintaan)

Request method menentukan karakteristik dari permintaan yang dikirimkan. Jika Anda adalah seorang *developer* web, Anda mungkin sudah mengenal dua metode yang umum, yaitu `GET` dan `POST`. Selain kedua metode tersebut, ada metode-metode lain yang dapat juga digunakan pada protokol HTTP seperti dijelaskan pada tabel berikut (tabel tidak mencakup seluruh metode).

Tabel 1.1. Daftar Metode HTTP

Metode	Deskripsi
`GET`	Metode yang paling umum digunakan, dan digunakan untuk mendapatkan konten dari *resource* yang ditentukan pada *request*.

Metode	Deskripsi
POST	Metode ini digunakan untuk meminta *server* memproses data yang dikirimkan. Pada umumnya, metode POST diikuti dengan *request body*, yang berisi parameter-parameter yang dikirimkan.
HEAD	Metode HEAD mirip dengan metode GET, tetapi bedanya di sini *server* tidak mengembalikan konten jawaban (*body*), melainkan hanya sampai *response headers* saja.
PUT	Metode ini digunakan untuk membuat atau menggantikan *resource* yang ditentukan pada *request*.
DELETE	Metode ini digunakan untuk menghapus sebuah *resource* dari *server*.

Dengan bervariasinya metode permintaan yang tersedia, bukan berarti semuanya diperkenankan oleh *server*. Pada *Apache Web Server* misalnya, secara *default* metode PUT dan DELETE tidak diperkenankan, sehingga pengguna umum tidak dapat seenaknya memanipulasi *resource* yang ada di *server*.

1.2.2. Resource (sumberdaya)

Standar RFC 7231 mendefinisikan *resource* sebagai target dari permintaan HTTP. *Resource* diidentifikasikan dengan sebuah *Uniform Resource Identifier* (URI), yang untuk protokol HTTP tersusun dari elemen-elemen berikut:

```
"http:" "//" authority path-abempty [ "?" query ] [ "#" fragment ]
```

Pada definisi di atas, *query* dan *fragment* bersifat opsional.

authority

Nama domain atau alamat IP dari *server* yang dituju. Dapat pula diikuti dengan tanda titik dua (":") dan nomer port yang digunakan, jika ingin menggunakan port selain *default* (80 untuk HTTP atau 443 untuk HTTPS). Contohnya: en.wikipedia.org, 198.35.26.96, atau google.com:8000.

path-abempty

Lokasi dari *resource* yang tersusun secara hirarkikal, seperti direktori. Disusun dengan tanda garis miring ("/") diikuti nama lokasi, untuk setiap tingkat. Elemen ini bisa juga kosong. Contohnya: /wiki/Indonesia, /search, /.

query

Menyatakan data non-hirarkikal, yang secara opsional disertakan pada permintaan. Pada metode GET, parameter-parameter biasanya disertakan pada elemen ini. Contohnya: q=Indonesia.

fragment

Menyatakan identifikasi tidak langsung pada sumberdaya sekunder. Pada *server* web, biasanya bagian ini tidak diterima oleh *server*, melainkan digunakan oleh peramban untuk mengidentifikasi *anchor* pada halaman. Contohnya: *Culture*

Berikut adalah contoh elemen-elemen di atas dalam URI lengkap:

- URI http://en.wikipedia.org/wiki/Indonesia#Culture memiliki elemen authority "en.wikipedia.org", path-abempty "*/wiki/Indonesia*", dan fragment "Culture"
- URI http://www.google.com/search?q=Indonesia memiliki elemen authority "www.google.com", path-abempty "/search", dan query "q=Indonesia".

URI digunakan sebagai *pengenal* sebuah sumberdaya. Anda mungkin juga pernah mendengar istilah URL (*Uniform Resource Locator*), yang digunakan sebagai *alamat* sebuah sumberdaya. Supaya sederhana, di buku ini kita asumsikan keduanya sama.

Pada *server* web seperti *Apache Web Server*, path-abempty biasanya diasosiasikan dengan file yang ada di *server*. Walaupun hal ini umum, tetapi standar RFC 7230 tidak mewajibkan hal ini. Sebagai contoh, gambar pada URI https://en.wikipedia.org/static/images/project-logos/enwiki.png bisa saja mengacu ke file "*enwiki.png*", atau bahkan tidak disimpan dalam bentuk file.

1.2.3. Request Headers

Request headers digunakan untuk memberikan informasi-informasi tambahan pada sebuah permintaan. Setiap *header* terdiri dari nama dan nilainya, terpisah oleh tanda titik dua dan spasi (": "). Tabel berikut menjelaskan beberapa *header* yang umum dipakai:

Tabel 1.2. Daftar Request Header yang Umum

Header	Deskripsi
`Host`	Menunjukkan alamat dari situs web yang ingin diakses, terutama berguna jika satu *server* melayani lebih dari satu situs. Pada protokol HTTP versi 1.1 ke atas, header ini wajib ada.
`Referer`	Menunjukkan alamat yang mereferensikan situs yang dituju. Sebagai contoh, jika Anda mencari "Wikipedia" di Google, dan mengklik tautannya, maka permintaan ke situs http://www.wikipedia.org akan memiliki header Referer berupa http://www.google.com.
`User-Agent`	Menunjukkan jenis peramban yang digunakan. Isinya adalah satu atau lebih nama peramban, dipisahkan oleh spasi. Header ini wajib ada, dan biasanya dimanfaatkan *server* untuk mengatasi masalah kompatibilitas situs saat dibuka dari berbagai peramban yang berbeda.

1.3. Format Jawaban

Data yang dikirimkan pada langkah 3 (jawaban / *response*) adalah seperti berikut:

```
HTTP/1.1 200 OK ❶
Content-Encoding: gzip ❷
Vary: Accept-Encoding
Last-Modified: Fri, 18 Dec 2015 22:42:10 GMT
Content-Type: text/html; charset=UTF-8
Cache-Control: public, max-age=3600
Content-Length: 1918
Accept-Ranges: bytes
Date: Tue, 16 Feb 2016 03:36:15 GMT
```

```
Server: Apache
X-Cache-Host: cache2.lax.icann.org
X-Cache-Hits: 36272

<!doctype html> ❸
<html>
<head>
 <title>Internet Assigned Numbers Authority</title>

 <meta charset="utf-8" />
 <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
 <meta name="viewport" content="width=device-width, initial-scale=1" />

 <link rel="stylesheet" media="screen" href="/_css/2015.1/screen.css"/
> ❹

 ...
```

❶ Baris pertama pada jawaban disebut *status line*, dan diawali diawali dengan versi protokol yang digunakan, dalam kasus ini `HTTP/1.1`. *Status line* diikuti dengan 3 dijit kode status, dalam kasus ini `200`. *Status line* diakhiri dengan representasi tekstual dari status tersebut, dalam kasus ini *OK*). Pada umumnya, representasi tekstual tersebut diabaikan oleh peramban dan hanya berfungsi untuk informasi saja.

❷ Baris kedua dan baris-baris berikutnya, sampai ditemukan baris kosong, berisi *response headers* dalam format "*nama-header: nilai-header*". Pada contoh di atas, terdapat *header* `Server` berisi *Apache* yang menandakan bahwa *server* yang digunakan untuk melayani permintaan adalah *Apache Web _Server*.

❸ Setelah baris kosong adalah *body* dari *response*, dalam kasus ini berupa teks HTML.

❹ Perhatikan bahwa ada kebutuhan akan *file* "screen.css" di HTML ini. *File* tersebut akan diunduh secara terpisah, tetapi juga dengan protokol HTTP.

1.3.1. Kode Status

Kode status adalah bilangan bulat tiga dijit yang menyatakan status dari pemrosesan permintaan yang dikirimkan. Berikut adalah beberapa kode status yang umum ditemui:

Tabel 1.3. Kode Status yang Umum Ditemui

Kode Status	Status	Deskripsi
200	OK	*Request* berhasil diproses dengan baik.
301	Moved Permanently	*Resource* yang diminta sudah berpindah ke URI yang lain secara permanen.
302	Found	*Resource* yang diminta untuk sementara berpindah pada URL yang lain. Untuk alasan historis, *client* diperkenankan untuk mengubah metode permintaan dari `POST` menjadi `GET` (fitur pada versi terdahulu).
307	Temporary Redirect	*Resource* yang diminta untuk sementara berpindah pada URL yang lain. Mirip dengan status `302`, namun *client* tidak diperkenankan mengubah metode permintaan dari `POST` menjadi `GET`.
400	Bad Request	*Server* tidak dapat memproses permintaan karena ada kesalahan dari *client* (misal: kesalahan sintaks, parameter, dll).
401	Unauthorized	*Server* tidak dapat memproses permintaan karena kredensial (*username*/*password*) diperlukan, dan *client* tidak menyediakannya. *Client* dapat mencoba mengirimkan lagi permintaan dengan menyediakan kredensial yang diperlukan.
403	Forbidden	*Server* menolak untuk memberikan jawaban, karena *client* tidak diperbolehkan mengakses sumberdaya yang diminta (bahkan dengan memberikan kredensial).
404	Not Found	*Resource* yang diminta tidak tersedia pada *server* (dapat juga, *server* menolak untuk menginformasikan status keberadaan sumberdaya yang diminta).
500	Internal *Server* Error	*Server* mengalami masalah internal, sehingga tidak dapat memproses permintaan yang dikirimkan.
501	Not Implemented	*Server* belum/tidak mendukung fungsionalitas yang diminta oleh *client*.

Kode Status	Status	Deskripsi
503	Service Unavailable	*Server* tidak dapat menjawab permintaan *client*, karena terlalu sibuk atau perawatan. Status ini mengindikasikan *client* dapat mencoba lagi setelah jangka waktu tertentu.

Kode status yang tersedia dikelompokkan menjadi lima, diindikasikan oleh dijit pertama dari kode tersebut:

- 1xx (Informational): *Request* diterima, dan proses dilanjutkan
- 2xx (Successful): *Request* diterima dan dimengerti dengan baik
- 3xx (Redirection): Aksi tambahan dipelukan untuk menyelesaikan permintaan
- 4xx (Client Error): Terjadi kesalahan dan *client* harus memperbaikinya
- 5xx (*Server* Error): Terjadi kesalahan pada sisi *server*

1.3.2. Response Headers

Response headers digunakan untuk memberikan informasi-informasi tambahan pada sebuah jawaban. Sama seperti *request header*, Setiap *header* terdiri dari nama dan nilai, dan terpisah oleh titik dua dan spasi ("*:* "). Tabel berikut menjelaskan beberapa *header* yang umum dipakai:

Tabel 1.4. Daftar Response Header yang Umum

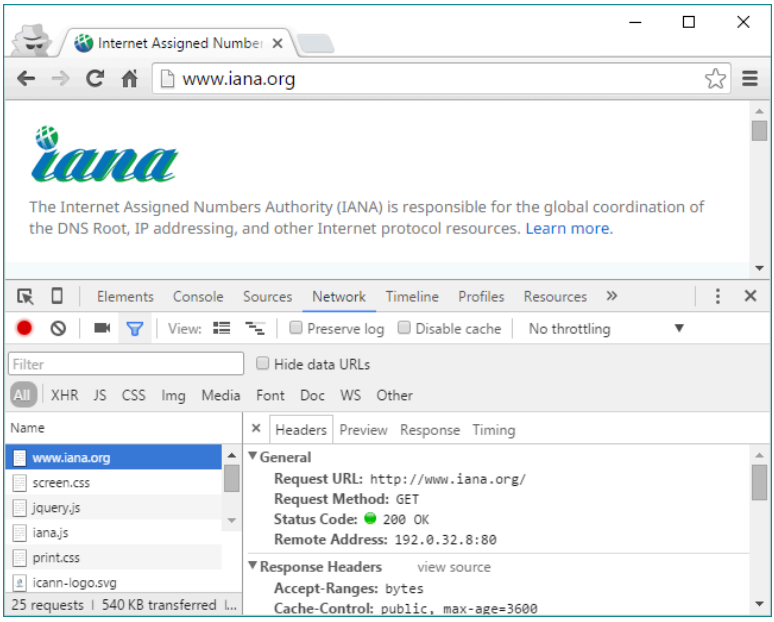
Header	Deskripsi
`Content-Type`	*Header* ini menunjukkan tipe media dari konten yang akan diberikan. Pada bentuk yang paling sederhana, nilai dari *header* ini berisi dari kode tipe MIME (*Multipurpose Internet Mail Extensions*). Beberapa kode tipe MIME yang umum antara lain: `text/plain` untuk teks, `text/html` untuk halaman HTML; `image/gif`, `image/jpeg`, dan `image/png` untuk gambar berformat GIF, JPEG, dan PNG; `application/json` untuk data JSON (akan dijelaskan pada bab berikutnya).

Header	Deskripsi
`Cache-Control`	*Header* ini mengatur bagaimana konten yang dikirimkan dapat dikirimkan sementara di *client*. Pada konten-konten statis seperti gambar, secara *default* konten akan disimpan di *client* dalam jangka waktu tertentu, sehingga jika dibutuhkan dalam waktu dekat di masa depan, tidak perlu mengirimkan permintaan lagi ke *server*. Jika secara eksplisit diinginkan konten diminta lagi setiap kali diperlukan, dapat mengisi *header* ini dengan nilai `no-cache`.
`Location`	*Header* ini digunakan untuk beberapa jenis jawaban untuk menunjukkan lokasi sumberdaya dalam bentuk URI. Pada jawaban dengan kode jawaban 3xx (contohnya: *301 Moved Permanently*, *302 Found*, atau *307 Temporary Redirect*), nilai dari header ini menunjukkan lokasi baru yang harus dituju.

1.4. Melihat Komunikasi HTTP

Cara termudah untuk melihat komunikasi yang terjadi antara *browser* dengan *server* web adalah dengan menggunakan fitur *Developer tools* yang tersedia di sebagian besar *browser* di komputer. Pada Google Chrome, *tools* ini dapat dimunculkan dengan menekan tombol Ctrl+Shift+I. Kemudian, Anda harus memilih *tab Network*.

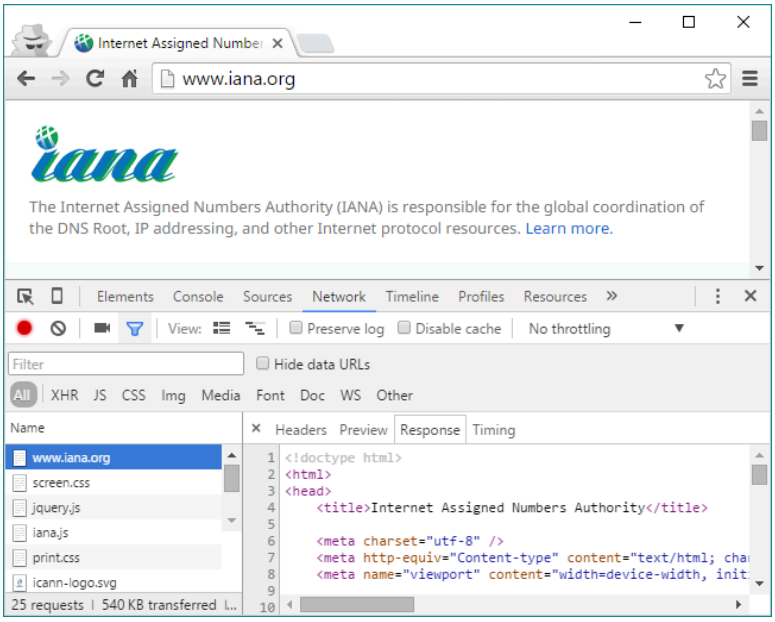
Jika *tab Network* sudah aktif, saat sebuah halaman dibuka, peramban akan menampilkan daftar *request* yang dilakukan. Saat salah satu *request* dipilih, peramban akan menampilkan detail dari *request* beserta jawabannya.



Gambar 1.3. Komunikasi Halaman Situs www.iana.org

Pada contoh gambar di atas, terlihat ada permintaan utama ke domain *www.iana.org*, diikuti dengan *resources* pendukung seperti `screen.css`, `jquery.js`, dan sebagainya. Pada subtab *Headers*, Anda dapat melihat isi dari *Response Headers* maupun *Request Headers* yang terjadi, seperti yang telah dijelaskan pada subbab-subbab sebelum ini.

Subtab *Preview* maupun *Response* keduanya menampilkan *response body*. Bedanya, subtab *Response* akan menampilkan hasil olahan dari tubuh tersebut jika berupa data multimedia (misal: menampilkan gambar jika tipe konten adalah `image/jpeg`). Gambar di bawah ini menampilkan tubuh dari jawaban atas permintaan ke http://www.iana.org.



Gambar 1.4. Subtab Response Halaman Situs http://www.iana.org

Terdapat dua subtab lagi, yaitu *Sources* dan *Timeline*. Keduanya tidak terlalu relevan dengan topik buku ini dan tidak akan dibahas lebih lanjut. Jika Anda ingin mempelajari lebih dalam mengenai *Developer Tools* Google Chrome, silahkan mengakses situs resminya [1].

1.5. Kesimpulan

Di bab ini, kita mempelajari apa yang terjadi di balik layar saat mengakses sebuah situs web. Komunikasi antara *client* dengan *server* memanfaatkan protokol HTTP. Protokol tersebut selalu terdiri dari *request* dari *client* yang diikuti dengan *response* dari *server*.

Kita juga belajar mengguakan *Chrome Developer Tools* untuk melihat komunikasi yang terjadi antara *client* dengan *server*.

1.6. Latihan

1. Identifikasikan nilai *authority*, *path-abempty*, *query*, & *fragment* dari URI-URI berikut:

[1] https://developers.google.com/web/tools/chrome-devtools/

   a. https://www.google.co.id
   b. https://www.google.co.id/search?q=webservice
   c. https://www.google.co.id/#q=webservice
   d. http://www.unpar.ac.id:80/peraturan-akademik
2. Apakah kode status HTTP yang tepat untuk skenario di bawah ini::
   a. Halaman tidak ditemukan di *server*.
   b. *Client* memberikan teks (misal: "abc") untuk parameter yang seharusnya berupa bilangan (misal: "harga=abc")
   c. URI berubah karena *rebranding* (misal www.tokobagus.com ⇒ www.olx.co.id)
   d. URI berubah karena ada *maintenance* di server utama (misal www.olx.co.id ⇒ www1.olx.co.id)
3. Cobalah untuk menangkap paket-paket yang dikirimkan dan diterima *browser* menggunakan aplikasi Wireshark [2]. Aplikasi ini mengangkap komunikasi pada level paket data, sehingga dapat membuktikan apakah protokol HTTP memang demikian seperti dijelaskan.
4. Cobalah melakukan hal yang sama dengan Wireshark, tetapi ujilah dengan mengakses situs yang diawali dengan https (misal: https://www.google.com). Apakah Anda dapat melihat protokol HTTP di sana? Mengapa?
5. Di bab ini, kita menggunakan *Developer Tools* milik *Google Chrome*. *Browser* lain seperti *Firefox* dan *Internet Explorer* pun memiliki *developer tools* masing-masing. Cobalah untuk menggunakannya!

[2] https://www.wireshark.org/

2

HTTP Sebagai Protokol Komunikasi

Karena aktivitas mengakses situs web biasanya dilakukan oleh aplikasi *browser*, aktivitas ini biasanya dianggap aman dan tidak terlalu berpotensi merusak sistem perangkat yang digunakan. Bandingkan dengan aplikasi seperti BitTorrent yang memerlukan aplikasi tertentu untuk di-*install* ke dalam sistem.

Adanya bermacam-macam aplikasi (selain *browser*) memungkinkan terjadinya celah keamanan, sehingga *system administrator* dari sebuah jaringan biasanya menghalangi komunikasi untuk *port-port* yang tidak umum. Dalam perkembangannya, *port* 80 dianggap cukup umum dan aman untuk tidak diblok. Bandingkan dengan protokol BitTorrent yang menggunakan *port* 6881-6889 dan 6969.

Karena *port* 80 umumnya tidak diblok, protokol HTTP menjadi salah satu protokol yang populer untuk digunakan. Keuntungan lainnya adalah protokol ini secara natif didukung oleh *browser*, sehingga komunikasi pun dapat dilakukan oleh aplikasi *client* berbasis web dengan *Javascript*.

Layanan *server* yang berkomunikasi menggunakan protokol HTTP disebut dengan "*web service*".

2.1. Studi Kasus: Daftar Belanja

Di sepanjang buku ini, kita akan menggunakan skenario aplikasi "Daftar Belanja" untuk studi kasus. Aplikasi ini secara sederhana mencatat barang-barang yang harus dibeli di supermarket, dan setiap barang memiliki penanda "cek" yang mencatat apakah barang tersebut sudah diambil atau belum.

Data barang belanjaan serta status "cek" nya disimpan di *server*, sedangkan *client* berfungsi sebagai antarmuka.

Gambar 2.1. Contoh Tampilan Aplikasi Daftar Belanja

2.2. Komunikasi Baca Saja

Pada subbab ini, kita akan mempelajari bagaimana memanfaatkan protokol HTTP untuk membaca data dari *server*.

2.2.1. HTTP dan HTML

Kita mulai dengan membuat aplikasi PHP sederhana. Tidak ada komunikasi *client-server* di sini, melainkan halaman langsung dibuat oleh skrip PHP yang dijalankan di *server*.

Dalam bahasa PHP, aplikasi tersebut diimplementasikan seperti berikut:

/daftar_html.php.

```
<!DOCTYPE html>
<html>
  <head><title>Daftar Belanja</title></head>
  <body>
    <h1>Daftar Belanja</h1>
    <ul>
<?php
    if (!file_exists('daftarbelanja.sqlite')) { ❶
      $db = new SQLite3('daftarbelanja.sqlite');
      $db->exec("CREATE TABLE daftarbelanja (nama TEXT PRIMARY KEY, cek
 INTEGER);");
```

```
      $db->exec("INSERT INTO daftarbelanja VALUES ('Tomat', 1);");
      $db->exec("INSERT INTO daftarbelanja VALUES ('Jeruk', 1);");
      $db->exec("INSERT INTO daftarbelanja VALUES ('Roti', 0);");
      $db->exec("INSERT INTO daftarbelanja VALUES ('Sabun Cuci', 0);");
      $db->exec("INSERT INTO daftarbelanja VALUES ('Sabun Mandi', 0);");
    } else {
      $db = new SQLite3('daftarbelanja.sqlite');
    }

    $results = $db->query('SELECT nama, cek FROM daftarbelanja'); ❷
    while ($row = $results->fetchArray()) {
      echo '      <li><input type="checkbox"' .
      ($row['cek'] === 0 ? '' : ' checked') .
      '>' .
      $row['nama'] .
      "</li>\n";
    }
?>
    </ul>
  </body>
</html>
```

Kode di atas memanfaatkan *library* SQLite3 pada PHP [1]. Jika Anda menggunakan PHP versi 5.3.0 ke atas, *library* ini sudah otomatis ada, sehingga Anda bisa langsung menggunakan.

❶ Kita memulai *script* PHP kita dengan memeriksa apakah file basis data "daftarbelanja.sqlite" sudah ada. Jika belum, kita buatkan file tersebut, dan mengisinya dengan data contoh. Jika sudah ada, kita tinggal membuka file tersebut.

❷ Setelah itu, kita lakukan *query* SELECT terhadap tabel *daftarbelanja* untuk mendapatkan barang-barang belanjaan kita. Daftar barang tersebut ditampilkan dalam bentuk HTML.

Kode di atas, jika ditaruh di sebuah web server (seperti *Apache Web Server* [2]) dan dibuka dari peramban akan menghasilkan tampilan seperti contoh tampilan daftar belanja di atas, yang dibentuk oleh kode HTML seperti di bawah ini:

[1] http://php.net/manual/en/book.sqlite3.php

[2] Salah satu cara mudah untuk menggunakannya adalah dengan menginstall aplikasi XAMPP (https://www.apachefriends.org)

```
<!DOCTYPE html>
<html>
  <head><title>Daftar Belanja</title></head>
  <body>
    <h1>Daftar Belanja</h1>
    <ul>
      <li><input type="checkbox" checked>Tomat</li>
      <li><input type="checkbox" checked>Jeruk</li>
      <li><input type="checkbox">Roti</li>
      <li><input type="checkbox">Sabun Cuci</li>
      <li><input type="checkbox">Sabun Mandi</li>
    </ul>
  </body>
</html>
```

Selain oleh *browser*, sebenarnya data tersebut juga bisa dibaca oleh program lain, termasuk program buatan kita sendiri. Contohnya, program Java di bawah mengirimkan permintaan ke alamat http://localhost/daftar.php dan mengolah hasil HTML nya untuk ditampilkan dengan format sendiri:

/chapter2/DaftarHTMLClient.java.

```
package chapter2;

import java.io.*;
import org.jsoup.*;
import org.jsoup.nodes.*;
import org.jsoup.select.*;

public class DaftarHTMLClient {

  public static void main(String[] args) throws IOException {
    Document document = Jsoup.connect("http://localhost/
daftar_html.php").get(); ❶
    Elements lists = document.select("li"); ❷
    for (Element list : lists) {
      Element checkbox = list.select("input").first(); ❸
      if (checkbox.hasAttr("checked")) { ❹
        System.out.println("[v] " + list.text());
      } else {
        System.out.println("[ ] " + list.text());
      }
    }
```

```
  }
}
```

Untuk alasan kesederhanaan, Kode di atas memanfaatkan *library* jsoup [3], yang membantu kita untuk mengirimkan *request* dan menerima *response* dari *server* HTTP serta mengolah HTML yang diberikan. Cara paling sederhana menggunakan *library* ini adalah menambahkan *file* `jsoup-x.x.x.jar` ke dalam *classpath* proyek Java Anda.

❶ Pertama-tama, kita kirimkan *request* HTTP ke http://localhost/daftar_html.php dengan metode `GET`. Hasilnya kita tangkap ke dalam variabel *document*.
❷ Setelah itu, kita cari *tag* `li` pada dokumen menggunakan *method* `select()`. Kita akan mendapatkan elemen-elemen `li` pada variabel "lists".
❸ Untuk setiap `li` yang didapat, kita cari *tag* `input` pertama di dalamnya (walaupun pasti ada tepat satu *tag* `input`).
❹ Terakhir, kita tampilkan teks dari elemen tersebut, memanfaatkan *method* `text()`. Kita juga dibantu dengan *method* `hasAttr()` untuk memeriksa apakah *tag* tersebut mengandung atribut *checked*.

Jika dijalankan, program di atas akan menghasilkan keluaran seperti berikut.

```
[V] Tomat
[V] Jeruk
[ ] Roti
[ ] Sabun Cuci
[ ] Sabun Mandi
```

2.2.2. HTTP dan Format Custom

Jika diperhatikan lebih lanjut, metode komunikasi di subbab sebelumnya bisa dioptimasi lagi. Bahasa HTML diperlukan jika ingin menampilkan hasilnya langsung ke *browser*, namun tidak diperlukan jika ingin ditampilkan oleh program Java. Kita dapat memangkas keluaran yang dihasilkan oleh kode PHP tersebut, menjadi seperti berikut:

/daftar_plain.php.

[3] http://jsoup.org/

```
<?php
header('Content-type: text/plain'); ❶
// Diasumsikan daftarbelanja.sqlite sudah ada dan terisi dengan data
 contoh
$db = new SQLite3('daftarbelanja.sqlite'); ❷

$results = $db->query('SELECT nama, cek FROM daftarbelanja');
while ($row = $results->fetchArray()) {
  echo ($row['cek'] === 0 ? 'x ' : 'v ') . $row['nama'] . "\n";
}
```

Kode di atas tidak sepanjang kode sebelumnya *daftar_html.php* karena kita mengasumsikan berkas basis data *daftarbelanja.sqlite* sudah tersedia. Jika berkas tersebut hilang, Anda bisa membuatnya lagi dengan memanggil http://localhost/daftar_html.php.

❶ Perhatikan bahwa kode PHP tersebut diawali dengan perintah `header('Content-type: text/plain')`. Perintah ini berguna untuk menambahkan *response header* `Content-type: text/plain` yang berfungsi untuk menyatakan bahwa tipe media yang akan dikembalikan berupa teks polos (*plaintext*). Pada kode "daftar_html.php" sebelumnya, tidak ada perintah serupa dan tipe media akan menggunakan tipe *default* dari *web server* yaitu `text/html`.

❷ Baris-baris berikutnya mirip dengan *daftar_html.php*. Perbedaannya hanyalah pada *daftar_plain.php*, *tag* HTML tidak dituliskan.

Jika dijalankan, kode tersebut akan menghasilkan keluaran seperti berikut:

```
v Tomat
v Jeruk
x Roti
x Sabun Cuci
x Sabun Mandi
```

Perhatikan bahwa keluaran yang diberikan "daftar_plain.php" tersebut jauh lebih ringkas dibandingkan keluaran "daftar_html.php". Keluaran ini tidak terlihat bagus di peramban, tetapi sudah cukup jika hasilnya akan diproses lagi oleh sebuah program, seperti pada kode di bawah ini.

/chapter2/DaftarPlainClient.java.

```
package chapter2;

import java.io.*;
import org.jsoup.*;
import org.jsoup.helper.*;

public class DaftarPlainClient {

  public static void main(String[] args) throws IOException {
    Connection connection = HttpConnection.connect("http://localhost/
daftar_plain.php"); ❶
    String content = connection.execute().body(); ❷
    String[] lines = content.split("\n"); ❸
    for (String line : lines) {
      if (line.charAt(0) == 'v') {
        System.out.println("[v] " + line.substring(2));
      } else {
        System.out.println("[ ] " + line.substring(2));
      }
    }
  }
}
```

❶ Di sini kita menyiapkan dulu variabel bertipe `Connection` untuk melakukan koneksi. Cara ini berbeda dengan "DaftarHTMLClient.java", karena data yang kita tangkap kali ini tidak dalam bentuk HTML dan tidak dapat ditampung pada kelas `Document`.

❷ Untuk mengirimkan *request* tersebut, kita panggil *method* `execute()`, dan langsung kita tangkap hasilnya dalam bentuk *string* dengan *method* `body()`.

❸ Akhirnya, kita proses *response* tersebut dengan memecahnya per baris dan membacanya sesuai format yang telah kita tetapkan.

Menggunakan format teks polos seperti di atas berguna untuk mengefisienkan pertukaran data, karena tidak mengandung informasi-informasi yang tidak berguna.

Dalam perkembangannya, muncul masalah baru, yaitu perbedaan format yang digunakan antara satu aplikasi dengan aplikasi lain. Sebagai contoh, keluaran dari "daftar_plain.php" menghasilkan keluaran di mana terdapat satu barang belanjaan per baris, masing-masing terdiri dari huruf "v" atau "x" diikuti dengan

spasi dan nama barang belanjaan. Bayangkan seorang *developer* lain membuat aplikasi *server* serupa, namun menggunakan format yang berbeda, di mana setiap baris berisi angka "1" atau "0", dan langsung diikuti dengan nama barang seperti berikut.

```
1Tomat
1Jeruk
0Roti
0Sabun Cuci
0Sabun Mandi
```

Dengan begitu, aplikasi *client* pun harus menyesuaikan dengan format yang baru. Jika aplikasi tersebut berkomunikasi dengan beberapa layanan dari sumber berbeda yang menggunakan format berbeda-beda, aplikasi tersebut harus mampu berkomunikasi dengan berbagai format data yang berbeda-beda pula!

Untuk mengatasi hal ini, muncul beberapa standar format data sehingga berbagai aplikasi dapat berkomunikasi dengan format data yang seragam. XML (*Extensible Markup Language*) merupakan format yang sempat cukup populer, dan menjadi cikal bakal bahasa HTML. Namun, XML memiliki kelemahan yang sama dengan HTML, yaitu terlalu banyaknya sintaks XML yang harus disisipkan ke dalam data yang dikomunikasikan, sehingga ukuran data menjadi besar. Berikut adalah contoh data yang sama dalam bentuk XML.

```
<items>
  <item cek="1">Tomat</item>
  <item cek="1">Jeruk</item>
  <item cek="0">Roti</item>
  <item cek="0">Sabun Cuci</item>
  <item cek="0">Sabun Mandi</item>
</items>
```

Format lainnya adalah JSON, yang lebih ringkas dan sederhana.

2.2.3. HTTP dan JSON

JSON (*Javascript Object Notation*) adalah format pertukaran data yang diturunkan dari bahasa ECMAScript (lebih dikenal dengan *JavaScript*). Bahasa JSON sendiri diatur dalam spesifikasi RFC7159 atau ECMA-404. Sebagai

contoh, data daftar barang belanjaan kita dapat dituliskan dengan sintaks JSON salah satunya seperti berikut:

```
{
  "status":"ok",
  "items":[
    {
      "cek":true,
      "nama":"Tomat"
    },
    {
      "cek":true,
      "nama":"Jeruk"
    },
    {
      "cek":false,
      "nama":"Roti"
    },
    {
      "cek":false,
      "nama":"Sabun Cuci"
    },
    {
      "cek":false,
      "nama":"Sabun Mandi"
    }
  ]
}
```

Perhatikan bahwa data di atas tidak seramping teks polos, namun juga tidak seboros HTML. Bagaimana cara membaca data tersebut? Sebelumnya, mari kita lihat dulu ringkasan spesifikasi bahasa JSON sesuai standar RFC 7159.

Sebuah nilai JSON bisa berupa salah satu dari 5 elemen berikut:

- **nilai literal**, yaitu salah satu dari "false", "null", atau "true".
- **bilangan**, berupa bilangan bulat maupun desimal, dengan tanda titik (".") sebagai pemisah antara bilangan bulat dengan pecahan.
- **string**, deretan karakter yang diapit dengan tanda kutip ganda (""""). Di dalamnya bisa terdapat karakter karakter khusus seperti *line feed* ("\n"), tanda kutip ganda secara literal ("\""), garis miring terbalik secara literal ("\\"), atau

kode *Unicode* dalam bentuk "\u*xxxx*" di mana *xxxx* adalah 4 dijit hekadesimal yang merepresentasikan kode *Unicode* yang dimaksud.

- **array**, nol atau lebih nilai JSON (yang disebut sebagai *elemen*) yang dipisahkan koma (",") dan diapit dengan kurung siku ("[" dan "]"). Masing-masing elemen dalam *array* tidak harus berjenis sama.
- **objek**, nol atau lebih *anggota* berupa pasangan nama dan nilai, yaitu *string* teks dan nilai JSON yang dipisahkan dengan tanda titik dua (":"). Untuk memisahkan anggota satu dengan lainnya, digunakan tanda koma (",").

Dari definisi tersebut, dapat kita simpulkan bahwa data JSON yang kita lihat sebelumnya adalah sebuah objek yang terdiri dari dua anggota, yaitu "status" dan "items". "status" selalu berisi "ok", yang memberitahukan kepada *client* bahwa permintaan daftar barang telah berhasil dilakukan. *items* merupakan *array* yang setiap elemennya merupakan objek yang mengandung dua anggota, *cek* dan *nama*. *cek* menyatakan apakah barang tersebut sudah dibeli, dan *nama* menyatakan nama barang. Walaupun JSON tidak mengatur bagaimana aplikasi harus memanfaatkan tipe-tipe data yang disediakan, tetapi ada baiknya untuk membuatnya seragam dalam sebuah aplikasi. Contohnya, kita akan selalu menggunakan anggota objek "status" untuk menentukan keberhasilan *request*.

Menulis dan membaca data JSON dapat dilakukan secara manual, namun biasanya akan lebih mudah dan terhindar dari kesalahan jika menggunakan *library* yang sudah ada. Pada bahasa PHP kita dapat memanfaatkan fungsi `json_encode()` [4] untuk mengkonversi variabel PHP menjadi sintaks JSON, seperti pada contoh kode program berikut:

/daftar_json.php.

```
<?php
header('Content-type: application/json'); ❶
// Diasumsikan daftarbelanja.sqlite sudah ada dan terisi dengan data
 contoh
$db = new SQLite3('daftarbelanja.sqlite'); ❷

$results = $db->query('SELECT nama, cek FROM daftarbelanja');
$json_result = array(
    'status' => 'ok',
```

[4] http://php.net/manual/en/book.json.php

```
    'items' => array()
);
while ($row = $results->fetchArray()) {
  $json_result['items'][] = array(
      'cek' => $row['cek'] === 1,
      'nama' => $row['nama']
  );
}
echo json_encode($json_result);
```

❶ Kali ini, kita menggunakan *header* `Content-type: application/json` yang menyatakan bahwa *response* dari layanan ini dalam format JSON.

❷ Baris-baris berikutnya mirip dengan "daftar_plain.php", tetapi dituliskan dalam format JSON dengan bantuan *function* `json_encode()`.

Jika Anda mencoba menjalankan program di atas, keluaran yang diberikan akan ditampilkan dalam satu baris saja. Standar JSON menyatakan bahwa *whitespace* (spasi dan kawan-kawannya) diabaikan kecuali di dalam *string*.

Perhatikan bahwa program di atas menuliskan *header* tipe konten berupa `application/json`, yang merupakan tipe resmi untuk format JSON.

Kita pun perlu mengubah kode Java kita untuk dapat membaca *response* dari *server* dalam format JSON. Di sini kita memanfaaatkan *library* JSON dasar [5]. *Library* ini tidak dibuat dalam bentuk JAR file, sehingga Anda perlu melakukan *copy-and-paste* kode Java nya ke proyek Anda.

Kode *client* yang dimaksud adalah seperti berikut:

/chapter2/DaftarJSONClient.java.

```
package chapter2;

import java.io.*;
import org.json.*;
import org.jsoup.*;
import org.jsoup.helper.*;

public class DaftarJSONClient {
```

[5] https://github.com/stleary/JSON-java

```
  public static void main(String[] args) throws IOException,
 JSONException {
    Connection connection = HttpConnection.connect("http://localhost/
daftar_json.php");
    connection.ignoreContentType(true); ❶
    String content = connection.execute().body();
    JSONObject jsonObject = new JSONObject(content); ❷
    JSONArray items = jsonObject.getJSONArray("items"); ❸
    for (int i = 0; i < items.length(); i++) { ❹
      JSONObject row = items.getJSONObject(i);
      if (row.getBoolean("cek")) {
        System.out.println("[v] " + row.getString("nama"));
      } else {
        System.out.println("[ ] " + row.getString("nama"));
      }
    }
  }
}
```

❶ Kode ini mirip dengan kode sebelumnya, hanya saja ditambahkan perintah `connection.ignoreContentType(true)` untuk memerintahkan *library* jsoup untuk mengabaikan tipe konten `application/json` yang dikirimkan pada *script* PHP. *Library* jsoup aslinya dirancang untuk menerima *response* berupa HTML, sehingga secara *default* menolak jika tipe konten bukan HTML atau teks polos.

❷ Untuk melakukan *parsing*, kita buat objek `JSONObject` dari *body response*.

❸ Dari situ, kita dapat mengakses anggota bernama *items*, dengan memanggil *method* `getJSONArray()`.

❹ Untuk setiap elemen *array*, kita dapatkan objeknya dengan memanfaatkan *method* `getJSONObject(index)`. Berikutnya, kita dapatkan nilai anggota *cek* dan *nama* menggunakan *method* `getBoolean()` dan `getString()`.

2.2.4. Parameter Permintaan

Pada contoh-contoh sebelumnya, kembalian dari *server* bersifat konstan, yaitu seluruh daftar barang yang tercatat. Ada kalanya, *client* perlu memberikan parameter-parameter tambahan yang dapat mempengaruhi hasil kembalian dari *server*. Sebagai contoh, misalkan kita ingin dapat menyaring daftar belanjaan kita agar mengembalikan barang-barang yang sudah dicek maupun tidak dicek saja.

Anda yang terbiasa dengan pemrograman PHP, pasti sudah mengenal *query string* yang merupakan parameter yang ada pada metode `GET`. Misalkan kita tetapkan bahwa *query string* "*checked*" dapat berisi nilai `true` atau `false`, yang menyatakan bahwa kita ingin mendapatkan barang-barang yang sudah dicek atau tidak dicek saja. Dengan begitu, kode program PHP kita akan menjadi seperti berikut.

/daftar_json_filter.php.

```
<?php

header('Content-type: application/json');
// Diasumsikan daftarbelanja.sqlite sudah ada dan terisi dengan data
 contoh
$db = new SQLite3('daftarbelanja.sqlite');

$query = 'SELECT nama, cek FROM daftarbelanja';
if (isset($_GET['checked'])) { ❶
  $query .= ' WHERE cek = ' . ($_GET['checked'] === 'true' ? '1' : '0');
}
$results = $db->query($query);
$json_result = array(
    'status' => 'ok',
    'items' => array()
);
while ($row = $results->fetchArray()) {
  $json_result['items'][] = array(
      'cek' => $row['cek'] === 1,
      'nama' => $row['nama']
  );
}
echo json_encode($json_result);
```

❶ Perbedaan kode ini dengan *daftar_json.php* adalah pada pemeriksaan parameter *checked* melalui *query string*. Jika parameter tersebut ditemukan, maka *SQL Query* yang kita bentuk akan ditambahkan *filter* yang memeriksa kolom *cek*.

Dengan demikian, permintaan pada alamat http://localhost/daftar_json_filter.php?checked=true akan mengembalikan:

```
{"status":"ok","items":[{"cek":true,"nama":"Tomat"},
{"cek":true,"nama":"Jeruk"}]}
```

Permintaan pada alamat http://localhost/daftar_json_filter.php?checked=true akan mengembalikan:

```
{"status":"ok","items":[{"cek":false,"nama":"Roti"},
{"cek":false,"nama":"Sabun Cuci"},{"cek":false,"nama":"Sabun Mandi"}]}
```

Menambahkan parameter "checked" pada aplikasi Java kita sangat mudah, yaitu hanya perlu menambahkan "checked=true" atau "checked=false" pada URL yang dituju. Jika Anda memilih untuk lebih terstruktur, dapat pula menggunakan *method* khusus `data(String, String)` yang milik jsoup.

/chapter2/DaftarJSONClientFilter.java.

```
package chapter2;

import java.io.*;
import org.json.*;
import org.jsoup.*;
import org.jsoup.helper.*;

public class DaftarJSONClientFilter {

  public static void main(String[] args) throws IOException,
 JSONException {
    Connection connection = HttpConnection.connect("http://localhost/
daftar_json_filter.php");
    connection.ignoreContentType(true);
    connection.data("checked", "true"); ❶
    String content = connection.execute().body();
    JSONObject jsonObject = new JSONObject(content);
    JSONArray items = jsonObject.getJSONArray("items");
    for (int i = 0; i < items.length(); i++) {
      JSONObject row = items.getJSONObject(i);
      if (row.getBoolean("cek")) {
        System.out.println("[v] " + row.getString("nama"));
      } else {
        System.out.println("[ ] " + row.getString("nama"));
      }
    }
  }
}
```

❶ Kode ini juga mirip dengan "DaftarJSONClient.java", hanya saja ditambahkan parameter "checked" bernilai "true". Supaya sederhana, parameter yang dikirimkan di sini di-*hardcode* pada program. Anda dapat mengubah isinya menjadi "false" untuk menampilkan barang-barang yang tidak dicek.

2.3. Komunikasi Baca-Tulis

Pada subbab sebelumnya, kita selalu melakukan operasi baca saja dari *server*. Memanfaatkan protokol HTTP, kita pun dapat melakukan operasi tulis ke *server*. Jika Anda perhatikan, seluruh contoh kode program yang sudah kita pelajari di bab ini secara implisit memanfaatkan metode GET dari protokol HTTP. Seperti dijelaskan pada bab 1, metode GET bukanlah satu-satunya metode yang didukung oleh HTTP. Ada metode lain, yaitu POST yang dapat kita gunakan untuk melakukan operasi tulis ke *server*. Metode ini lebih cocok untuk digunakan untuk menulis data dibandingkan dengan GET, antara lain karena:

1. Metode GET bersifat *cacheable*, yang artinya permintaan yang kedua kalinya bisa saja tidak diteruskan ke *server* jika *client* sudah memiliki data yang dibutuhkan. Dalam sebagian besar kasus, permintaan tulis harus selalu diteruskan ke *server*.
2. Metode POST dapat menampung lebih banyak data, karena parameter metode POST disimpan pada tubuh permintaan. Pada metode GET, parameter disimpan pada URL.

Di awal perkembangannya, metode POST dikatakan lebih aman terutama dalam mengirimkan *password*, karena *password* tidak akan terlihat di URL. Walaupun demikian, sesungguhnya metode POST sama tidak amannya dengan GET dalam mengirimkan *password*, karena keduanya mengirimkan *password* tanpa dienkripsi. Jika menginginkan pengiriman *password* yang aman, Anda perlu menggunakan protokol HTTPS.

2.3.1. Menambah Barang Baru

Sebagai contoh skenario operasi tulis, marilah kita implementasikan perintah untuk menambahkan barang belanjaan baru. Kita tetapkan sebuah parameter, yaitu "nama" yang berisi nama dari barang yang akan ditambahkan.

Ada beberapa cara untuk menyampaikan parameter ini ke server. Salah satunya adalah dengan menyertakannya dalam bentuk "name=value"pada bagian *body* dari permintaan HTTP, atau dikenal juga dengan "form-data". Format inilah yang digunakan jika kita menggunakan *tag* `form` pada halaman HTML, seperti pada potongan kode berikut:

/daftar_form_add.html.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Add Barang</title>
  </head>
  <body>
    <form method="POST" action="daftar_json_add.php">
      <label for="nama">Nama Barang:</label>
      <input type="text" id="nama" name="nama">
      <input type="submit"/>
    </form>
  </body>
</html>
```

Jika kode tersebut dibuka pada peramban dan diisi, maka akan mengirimkan permintaan POST ke "daftar_json_add.php" dengan kedua parameter yang telah kita tetapkan sebelumnya. Menangani perintah tersebut di *server* cukup sederhana, yaitu dengan membaca variabel `$_POST` dan menyisipkannya ke basis data. Yang perlu kita perhatikan adalah format _response_ dari perintah tersebut, yang sebaiknya juga menggunakan format JSON. Jika kode tersebut dibuka pada peramban dan diisi, maka akan mengirimkan permintaan POST ke "daftar_json_add.php" dengan kedua parameter yang telah kita tetapkan sebelumnya. Menangani perintah tersebut di *server* cukup sederhana, yaitu dengan membaca variabel `$_POST` dan menyisipkannya ke basis data. Yang perlu kita perhatikan adalah format _response_ dari perintah tersebut, yang sebaiknya juga menggunakan format JSON.

/daftar_json_add.php.

```
<?php
error_reporting(E_ALL - E_WARNING); ❶
header('Content-type: application/json');
// Diasumsikan daftarbelanja.sqlite sudah ada
$db = new SQLite3('daftarbelanja.sqlite');

$nama = $_POST['nama'];
if ($db->query("INSERT INTO daftarbelanja (nama, cek) VALUES('$nama',
 0)")) { ❷
  echo json_encode(array('status' => 'ok'));
} else {
  echo json_encode(array('status' => 'error', 'message' => $db-
>lastErrorMsg()));
}
```

❶ Kode program di atas diawali dengan mematikan fitur peringatan pada PHP. Saat menambahkan barang baru ke database, jika barang sudah ada PHP akan menampilkan pesan peringatan, yang dapat mengganggu format JSON yang kita kembalikan. Sebagai gantinya, kita akan mengembalikan JSON dengan "status" berisi "error" jika hal tersebut terjadi.

❷ Di sini kita melakukan perintah `INSERT` ke basis data, dan memeriksa hasilnya. Jika berhasil, mengembalikan objek JSON dengan nilai "status" berisi "ok". Jika gagal, mengembalikan objek JSON dengan nilai "status" berisi "error" dan "message" berisi pesan kesalahan.

Akhirnya, berikut adalah kode program *client* untuk mengirimkan perintah tambah barang tersebut.

/chapter2.DaftarJSONClientAdd.java.

```
package chapter2;

import java.io.*;
import java.util.*;
import org.json.*;
import org.jsoup.*;
import org.jsoup.Connection.Method;
import org.jsoup.helper.*;

public class DaftarJSONClientAdd {
```

```
  public static void main(String[] args) throws IOException,
 JSONException {
    Scanner scanner = new Scanner(System.in);
    System.out.print("Masukkan nama barang yang ingin ditambahkan: ");
    String nama = scanner.nextLine();
    Connection connection = HttpConnection.connect("http://localhost/
daftar_json_add.php");
    connection.ignoreContentType(true);
    connection.method(Method.POST); ❶
    connection.data("nama", nama);
    String content = connection.execute().body();
    JSONObject jsonObject = new JSONObject(content);
    String status = jsonObject.getString("status");
    if (status.equals("ok")) {
      System.out.println("Berhasil menambahkan " + nama);
    } else {
      System.out.println("Gagal menambahkan: " +
 jsonObject.getString("message"));
    }
  }
}
```

❶ Kode *client* ini juga mirip dengan kode sebelumnya. Hanya saja di sini kita mengubah metode pengiriman menjadi `POST` dan mengisikan parameter "nama" dengan nama barang yang ingin ditambahkan.

2.3.2. Membalik (Toggle) Status Cek Barang Belanjaan

Untuk melengkapi aplikasi kita, kita juga mengimplementasikan fitur membalik status barang belanjaan, dari tidak tercek menjadi tercek, atau sebaliknya. Sama seperti menambahkan barang, kita juga memerlukan satu buah parameter "nama" yang mendefinisikan nama barang yang ingin dibalik status cek nya.

Pada kasus ini, *client* juga perlu mengetahui status cek dari barang setelah diubah, sehingga dapat menampilkannya ke pengguna. Untuk itu, kita tetapkan kembalian dari *server* berupa variabel *item* yang berisi data barang setelah perubahan.

Berikut adalah kode pada *server* untuk memproses perubahan status cek:

/daftar_json_toggle.php.

```
<?php
```

```
error_reporting(E_ALL - E_WARNING);
header('Content-type: application/json');
// Diasumsikan daftarbelanja.sqlite sudah ada
$db = new SQLite3('daftarbelanja.sqlite');

$nama = $_POST['nama'];
if ($db->query("UPDATE daftarbelanja SET cek=1-cek WHERE nama='$nama'"))
 { ❶
  if ($db->changes() === 0) { ❷
    echo json_encode(array('status' => 'error', 'message' => "$nama
 tidak ditemukan"));
  } else { ❸
    $results = $db->query("SELECT nama, cek FROM daftarbelanja WHERE
 nama='$nama'");
    $row = $results->fetchArray();
    $item = array(
      'cek' => $row['cek'] === 1,
      'nama' => $row['nama']
    );
    echo json_encode(array('status' => 'ok', 'item' => $item));
  }
} else {
  echo json_encode(array('status' => 'error', 'message' => $db-
>lastErrorMsg()));
}
```

❶ Program yang kita buat mengirimkan perintah UPDATE ke tabel "daftarbelanja", di mana kita mengisikan nilai "cek" dengan kebalikannya ("1-cek").

❷ Setelah itu, kita cek juga apakah ada barang di tabel yang berubah akibat perintah 'UPDATE` tadi. Jika tidak ada, berarti kita tidak menemukan barang yang dimaksud dan kita kembalikan pesan kesalahan.

❸ Jika ada barang yang berubah, berarti kita telah berhasil melakukan pembalikan, dan kita kembalikan informasi barang tersebut dalam variabel "item".

Jika Anda ingin mengujinya dari *browser*, Anda dapat mengubah sedikit file "daftar_form_add.html" sehingga atribut "action" dari form mengarah ke "daftar_json_toggle.php". Di buku ini kita langsung akan membahas kode program *client* Java untuk mengirimkan perintah tersebut:

/chapter2/DaftarJSONClientToggle.java.

```
package chapter2;

import java.io.*;
import java.util.*;
import org.json.*;
import org.jsoup.*;
import org.jsoup.Connection.Method;
import org.jsoup.helper.*;

public class DaftarJSONClientToggle {

  public static void main(String[] args) throws IOException,
 JSONException {
    Scanner scanner = new Scanner(System.in);
    System.out.print("Masukkan nama barang yang ingin dibalik status cek
 nya: ");
    String nama = scanner.nextLine();
    Connection connection = HttpConnection.connect("http://localhost/
daftar_json_toggle.php");
    connection.ignoreContentType(true);
    connection.method(Method.POST); ❶
    connection.data("nama", nama);
    String content = connection.execute().body();
    JSONObject jsonObject = new JSONObject(content);
    String status = jsonObject.getString("status");
    if (status.equals("ok")) { ❷
      JSONObject item = jsonObject.getJSONObject("item");
      boolean cek = item.getBoolean("cek");
      System.out.println("Status cek dari " + nama + " sekarang adalah "
 + cek);
    } else {
      System.out.println("Gagal mengubah status " + nama + ": " +
 jsonObject.getString("message"));
    }
  }
}
```

❶ Tidak banyak yang berubah dari program kita. Kita tetap menggunakan metode POST dan memberikan parameter "nama".

❷ Setelah mengirimkan perintah, kita menuliskan kembali barang yang telah kita balikkan jika berhasil, atau menampilkan pesan kesalahan jika gagal.

2.3.3. Menghapus Barang Belanjaan

Untuk melengkapi, marilah kita buat fitur terakhir dari aplikasi daftar belanja kita, yaitu menghapus barang belanjaan. Mengikuti fitur-fitur sebelumnya, kita tetapkan satu buah parameter "nama" yang mendefinisikan nama barang yang ingin dihapus.

Berikut adalah kode pada *server* untuk menghapus barang belanjaan:

/daftar_json_delete.php.

```
<?php
error_reporting(E_ALL - E_WARNING);
header('Content-type: application/json');
// Diasumsikan daftarbelanja.sqlite sudah ada
$db = new SQLite3('daftarbelanja.sqlite');

$nama = $_POST['nama'];
if ($db->query("DELETE FROM daftarbelanja WHERE nama='$nama'")) { ❶
  if ($db->changes() === 0) {
    echo json_encode(array('status' => 'error', 'message' => "$nama
 tidak ditemukan"));
  } else {
    echo json_encode(array('status' => 'ok'));
  }
} else {
  echo json_encode(array('status' => 'error', 'message' => $db-
>lastErrorMsg()));
}
```

❶ Satu-satunya perbedaan adalah bahwa kode ini mengirimkan perintah `DELETE` pada SQL.

Serupa dengan kode-kode *client* sebelumnya, berikut adalah kode untuk mengirimkan perintah hapus:

/chapter2/DaftarJSONClientDelete.java.

```
import java.io.*;
import java.util.*;
import org.json.*;
import org.jsoup.*;
import org.jsoup.Connection.Method;
import org.jsoup.helper.*;
```

```
public class DaftarJSONClientDelete {

  public static void main(String[] args) throws IOException,
 JSONException {
    Scanner scanner = new Scanner(System.in);
    System.out.print("Masukkan nama barang yang ingin dihapus: ");
    String nama = scanner.nextLine();
    Connection connection = HttpConnection.connect("http://localhost/
daftar_json_delete.php");
    connection.ignoreContentType(true);
    connection.method(Method.POST);
    connection.data("nama", nama);
    String content = connection.execute().body();
    JSONObject jsonObject = new JSONObject(content);
    String status = jsonObject.getString("status");
    if (status.equals("ok")) {
      System.out.println("Berhasil menghapus " + nama);
    } else {
      System.out.println("Gagal menghapus: " +
 jsonObject.getString("message"));
    }
  }
}
```

2.4. Kesimpulan

Di subbab ini, kita telah membuat beberapa *script* PHP untuk memanipulasi daftar belanjaan. Secara kolektif, kita dapat menyebut *script-script* tersebut sebagai *web service*, karena *script* kita melayani *requests* untuk manipulasi melalui protokol HTTP yang digunakan di *web*. Di bab berikutnya, kita akan merapihkan *script-script* kita dengan gaya arsitektur REST.

2.5. Latihan

1. Punya dua komputer? Cobalah jalankan *server* dan *client* di komputer berbeda, dan pastikan mereka dapat berkomunikasi!
2. Kode Java di bab ini memanfaatkan *library* jsoup untuk melakukan *request* dengan protokol HTTP. Ada banyak *library* lainnya yang dapat digunakan,

seperti milik Java sendiri [6] atau Apache HTTP Components [7]. Perluas wawasan Anda dengan mencoba *library-library* tersebut!

3. Sama halnya dengan *library* JSON, cobalah pelajari cara untuk menggunakan *library* lain yang cukup populer, yaitu Jackson [8].

[6] https://docs.oracle.com/javase/tutorial/networking/urls/readingWriting.html
[7] https://hc.apache.org/httpcomponents-client-ga/
[8] https://github.com/FasterXML/jackson

3

Web Service yang Rapi

Di bab sebelumnya, Anda telah berhasil membangun sebuah *web service* untuk memanipulasi daftar belanjaan. Di bab ini, kita akan merapihkan *web service* yang kita miliki, sehingga lebih mudah digunakan dan efisien. Kita akan membahas dulu faktor-faktor apa yang membuat sebuah *web service* lebih rapi, dan di subbab terakhir kita akan mengimplementasikannya ke dalam *web service* Daftar Belanja kita.

3.1. RESTful *Web Service*

REST adalah gaya arsitektur yang dicetuskan pertama kali oleh Roy T. Fielding pada desertasinya yang berjudul "*Architectural Styles and the Design of Network-based Software Architectures*". Walaupun bisa diaplikasikan pada berbagai protokol, tetapi HTTP (web) telah menjadi salah satu protokol yang cocok untuk mengimplementasikan arsitektur ini.

Karakerisitik arsitektur REST ditentukan oleh beberapa *constraints* (batasan). Di bab ini, kita akan membahas beberapa *constraint* yang penulis rasa cukup penting.

3.1.1. Client-Server

Aplikasi berarsitektur REST bersifat *client-server*, di mana sebagian pemrosesan dan penyimpanan data berada di *server*, sedangkan *client* biasanya berfungsi sebagai antarmuka bagi pengguna. Hal ini memungkinkan pemisahan tanggung jawab, dan dalam jangka panjang memudahkan pengembangan. Aplikasi daftar belanjaan kita sesungguhnya sudah menerapkan prinsip *client-server* ini, di mana

penyimpanan dan pengolahan data ditangani oleh *script* PHP yang ada di *server*, dan aplikasi Java sederhana kita berfungsi sebagai antarmuka.

Menariknya, dalam protokol HTTP sebuah entitas dapat berperan sebagai *client* dan *server* sekaligus, seperti didefinisikan pada standar RFC 7230 berikut:

> The terms "client" and "server" refer only to the roles that these programs perform for a particular connection. The same program might act as a client on some connections and a server on others. The term "user agent" refers to any of the various client programs that initiate a request, including (but not limited to) browsers, spiders (web-based robots), command-line tools, custom applications, and mobile apps. The term "origin server" refers to the program that can originate authoritative responses for a given target resource. The terms "sender" and "recipient" refer to any implementation that sends or receives a given message, respectively.

Entitas yang berfungsi sebagai *client* dan *server* sekaligus banyak ditemui pada aplikasi dengan arsitektur *SOA* (*Service Oriented Architecture*). Contohnya, sebuah *server* yang melayani manajemen penggajian dalam sebuah perusahaan, secara rutin membutuhkan layanan dari *server* lain (baca: menjadi *client*) yang melayani manajemen karyawan.

3.1.2. Stateless

Setiap *request* pada *web service* berarsitektur REST bersifat *stateless*, yang berarti *request* tersebut tidak boleh bergantung pada *request* sebelumnya. Dengan kata lain, sebuah *request* harus mengandung semua informasi yang dibutuhkan untuk mendapatkan *response* yang tepat.

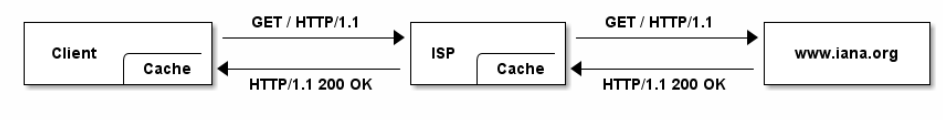
Protokol HTTP secara alami sudah bersifat *stateless*, sehingga kita tidak perlu terlalu khawatir terhadap *constraint* ini. Yang perlu diperhatikan hanyalah untuk menghindari penggunaan *cookie*, karena sifatnya yang *stateful* (sekedar pengingat, *cookie* digunakan untuk menyimpan *state* pada *browser*, untuk digunakan pada *request-request* berikutnya).

Seperti dibahas pada desertasi Roy T. Fielding, keuntungan dari sifat *stateless* ini adalah "*visibility, reliability, and scalability*". Kita mendapatkan keuntungan *visibility* karena kita cukup menganalisa satu *request* saja untuk memahami maksudnya. Keuntungan *reliability* didapat karena lebih mudah untuk pulih dari masalah (*recover*), karena tidak bergantung pada *request* sebelumnya. Keuntungan *scalability* didapat karena dengan tidak menyimpan *state* pada *request*, kita tidak membutuhkan ruang memori yang terlalu banyak pada *server*.

Jika Anda perhatikan, *web service* daftar belanja kita sudah memenuhi *constraint stateless*.

3.1.3. Cacheable

Protokol berarsitektur REST mendukung sistem *cache*, yang berarti *response* dari *request* yang sama dapat disimpan pada *client*, atau pada penghubung antara *client* dengan *server* untuk mempercepat respon serta mengurangi beban jaringan. Sebagai contoh, pada gambar di bawah ini, komunikasi antara *client* dengan *www.iana.org* dilakukan melewati ISP (*Internet Service Provider*) seperti Telkom atau Indosat. Baik *client* maupun ISP dapat menyimpan *response* yang dikirimkan dari *www.iana.org* sehingga pada *request* berikutnya, tidak perlu diteruskan ke *www.iana.org*.



Gambar 3.1. Komunikasi dengan www.iana.org melalui ISP

Standar RFC 7231 menetapkan metode GET, HEAD, dan (dengan syarat tertentu) POST sebagai *cacheable* yang berarti *client* maupun perantara dapat menyimpan *response* dari *request* yang sama. Walaupun demikian, sebagian besar implementasi *server* tidak menganggap metode POST sebagai *cacheable*.

Web service daftar belanja kita menggunakan GET untuk melakukan operasi baca dan POST untuk operasi tulis. Operasi baca kita terbilang cukup statis (hasilnya

tidak berubah-ubah), sehingga aman jika *response* disimpan dalam *cache*. Semua operasi tulis kita menggunakan `POST` yang tidak *cacheable*, sehingga kita lebih yakin bahwa setiap operasi tulis akan diteruskan ke *server*.

3.1.4. Uniform Interface

Constraint lain dari arsitektur REST adalah antarmuka protokolnya yang diatur seragam. Ada 4 (sub-)*constraint* yang diatur pada *constraint* ini, yaitu:

1. **Identification of Resources** Seperti pada konsep pemrograman berorientasi objek, kita menentukan *resource* apa saja yang ada pada *web service* kita, dan manipulasi dilakukan berdasarkan *resource* yang sudah diidentifikasi tersebut. Sebagai contoh, untuk daftar belanja kita tentukan *resource*-nya sebagai "barang" (karena kita ingin memanipulasi barang belanjaan), sehingga URI yang tepat adalah http://localhost/barang.
2. **Manipulation of Resources Through Representations** Idealnya, sebuah *RESTful Web Service* dapat melayani manipulasi data dengan memanfaatkan URI yang sama. Jika menggunakan protokol HTTP, ini berarti menggunakan metode `PUT` untuk menambahkan barang baru, `GET` untuk mendapatkan informasi barang, `POST` untuk memodifikasi barang, dan `DELETE` untuk menghapus barang. Di pembahasan buku ini, seluruh operasi modifikasi dan hapus menggunhakan metode `POST` supaya sederhana, tetapi Anda dapat mencoba metode `PUT` dan `DELETE` sebagai latihan.
3. **Self-descriptive Messages** *Response* yang diberikan oleh *server* sebaiknya memiliki informasi tentang bagaimana mengolah konten yang diberikan. Pada protokol HTTP, kita memanfaatkan header `Content-type`. Contohnya, pada *web service* Daftar Belanja, kita menggunakan tipe konten `application/json`. Untuk *response* berupa gambar, kita dapat menggunakan tipe `image/gif`, `image/jpeg`, `image/png`, atau tipe-tipe gambar lainnya.
4. **Hypermedia as the Engine of Application State** Secara sederhana, pada *web service* kita memanfaatkan URI untuk menunjukkan layanan-layanan yang tersedia. Contohnya, URI http://localhost/barang dapat kita sisipkan pada *response* dari layanan-layanan tertentu, untuk mengacu ke *resource* daftar barang. Selain itu, URI http://localhost/barang/Tomat dapat kita gunakan untuk mengacu ke *resource* barang bernama "Tomat".

3.2. *Status Code*

Protokol HTTP memiliki beberapa *status code* yang dapat digunakan untuk memberitahukan ke *client*, status dari *request* yang diberikan. Beberapa *status code* yang dapat kita gunakan, antara lain:

- 200 (*OK*): *Status code default* yang dikembalikan oleh PHP, jika kita tidak secara eksplisit mengubahnya. *Status code* ini kita gunakan untuk seluruh *request* yang berhasil dilaksanakan.
- 201 (*Created*): *Status code* yang menyatakan bahwa sebuah *resource* berhasil dibuat. Sesuai standar RFC 7231, *response* harus mengandung *response header* `Location` yang berisi URI dari *resource* yang baru saja dibuat.
- 400 (*Bad Request*): Kita dapat menggunakan kode status ini jika ada parameter pada *request* yang tidak sesuai yang *server* harapkan, misalnya jika *request* meminta menambahkan barang baru tetapi nama barang (parameter "nama") tidak disediakan.
- 404 (*Not Found*): Biasanya secara otomatis dikembalikan oleh *web server* seperti *Apache*, tetapi bisa juga kita gunakan jika ada *resource* yang secara eksplisit kita nyatakan tidak ada.
- 500 (*Internal Server Error*): Kita dapat menggunakan kode ini untuk memberitahukan kepada *client* bahwa ada kesalahan pada *server* yang tidak diharapkan, dan tidak terkait dengan parameter yang dikirimkan *client*.

Walaupun informasi mengenai status *request* juga dapat dikirimkan melalui *body* dalam format JSON, menginformasikannya melalui *status code* memiliki keuntungan tambahan. Informasi pada *status code* memungkinkan *client* secara cepat memutuskan apakah *request* yang dikirimkan berhasil atau tidak, tanpa harus mengurai JSON pada *body*. Hal ini terutama bermanfaat pada perangkat bergerak, di mana semakin banyak proses yang dilakukan akan menghabiskan baterai lebih banyak pula.

3.3. Implementasi pada Daftar Belanja

Untuk mendukung *requirement* REST "*Uniform Interface*", kita perlu menata ulang URI kita untuk *web service* daftar belanjaan. Kita tetapkan aturan sebagai berikut:

1. GET http://localhost/barang/ untuk mendapatkan daftar barang belanjaan
2. GET http://localhost/barang/nama-barang untuk mendapatkan detail barang belanjaan
3. POST http://localhost/barang/ dengan parameter `mode=add` untuk menambahkan barang belanjaan baru
4. POST http://localhost/barang/namabarang dengan parameter `mode=delete` untuk menghapus sebuah barang belanjaan
5. POST http://localhost/barang/nama-barang dengan parameter `mode=toggle` untuk membalik status cek barang belanjaan.

Selain itu, kita juga akan mengimplementasikan *status code* yang sesuai, untuk setiap layanan yang kita buat:

- 200 untuk setiap operasi yang berhasil
- 201 untuk operasi "add" yang berhasil
- 404 jika operasi atau barang tidak ditemukan
- 500 jika ada kesalahan pada basis data

> Penulis menemukan masalah terkait ada atau tidaknya karakter garis miring ("/") di akhir URI, terutama pada operasi menambahkan barang. Solusi sederhananya adalah mencoba menambahkan atau menghilangkan karakter tersebut jika terjadi masalah. Masalah ini terkait dengan standar penamaan direktori serta implementasi *web server* yang berbeda, dan untuk mempelajari lebih lanjut, Anda dapat mencari "*URI trailing slash*" di Google.

Pada *server Apache*, untuk mengimplementasikan fitur-fitur tersebut, kita cukup membuat file `index.php` yang berada dalam direktori `barang/`, dengan kode berikut:

/barang/index.php.

```
<?php

error_reporting(E_ALL - E_WARNING);
$db = getOrCreateDatabase();
```

```
$status_code = 404; ❶
$json_result = array(
  'status' => 'error',
  'message' => 'Layanan tidak ditemukan'
);

if ($_SERVER['REQUEST_METHOD'] === 'GET') { ❷
  if (!isset($_SERVER['PATH_INFO'])) { ❸
    $results = $db->query('SELECT nama, cek FROM daftarbelanja');
    $json_result = array(
      'status' => 'ok',
      'items' => array()
    );
    while ($row = $results->fetchArray()) {
      $json_result['items'][] = array(
        'cek' => $row['cek'] === 1,
        'nama' => $row['nama']
      );
    }
    $status_code = 200;
  } else { ❹
    $nama = substr($_SERVER['PATH_INFO'], 1);
    $results = $db->query("SELECT nama, cek FROM daftarbelanja WHERE
 nama='$nama'");
    $row = $results->fetchArray();
    if ($row === FALSE) {
      $json_result = array(
        'status' => 'error',
        'message' => "Barang $nama tidak ditemukan!"
      );
      $status_code = 404;
    } else {
      $json_result = array(
        'status' => 'ok',
        'item' => array(
          'nama' => $row['nama'],
          'cek' => $row['cek'] === 1
        )
      );
      $status_code = 200;
    }
  }
} else if ($_SERVER['REQUEST_METHOD'] === 'POST'
 && isset($_POST['mode'])) { ❺
```

```
 switch ($_POST['mode']) {
   case 'add': ❻
     $nama = $_POST['nama'];
     if ($db->query("INSERT INTO daftarbelanja (nama, cek)
VALUES('$nama', 0)")) {
       $json_result = array('status' => 'ok');
       $status_code = 201;
       header("Location: /barang/$nama");
     } else {
       $json_result = array(
         'status' => 'error',
         'message' => $db->lastErrorMsg()
       );
       $status_code = 500;
     }
     break;
   case 'delete': ❼
     $nama = substr($_SERVER['PATH_INFO'], 1);
     if ($db->query("DELETE FROM daftarbelanja WHERE nama='$nama'")) {
       if ($db->changes() === 0) {
         $json_result = array(
           'status' => 'error',
           'message' => "$nama tidak ditemukan"
         );
         $status_code = 404;
       } else {
         $json_result = array('status' => 'ok');
         $status_code = 200;
       }
     } else {
       $json_result = array(
         'status' => 'error',
         'message' => $db->lastErrorMsg()
       );
       $status_code = 500;
     }
     break;
   case 'toggle': ❽
     $nama = substr($_SERVER['PATH_INFO'], 1);
     if ($db->query("UPDATE daftarbelanja SET cek=1-cek WHERE
nama='$nama'")) {
       if ($db->changes() === 0) {
         $json_result = array(
           'status' => 'error',
           'message' => "$nama tidak ditemukan",
```

```
          );
          $status_code = 404;
        } else {
          $results = $db->query("SELECT nama, cek FROM daftarbelanja
 WHERE nama='$nama'");
          $row = $results->fetchArray();
          $json_result = array(
            'status' => 'ok',
            'item' => array(
              'cek' => $row['cek'] === 1,
              'nama' => $row['nama']
            )
          );
          $status_code = 200;
        }
      } else {
        $json_result = array(
          'status' => 'error',
          'message' => $db->lastErrorMsg()
        );
        $status_code = 500;
      }
      break;
  }
}
http_response_code($status_code);
echo json_encode($json_result);

function getOrCreateDatabase() {
  if (!file_exists('daftarbelanja.sqlite')) {
    $db = new SQLite3('daftarbelanja.sqlite');
    $db->exec("CREATE TABLE daftarbelanja (nama TEXT PRIMARY KEY, cek
 INTEGER);");
    $db->exec("INSERT INTO daftarbelanja VALUES ('Tomat', 1);");
    $db->exec("INSERT INTO daftarbelanja VALUES ('Jeruk', 1);");
    $db->exec("INSERT INTO daftarbelanja VALUES ('Roti', 0);");
    $db->exec("INSERT INTO daftarbelanja VALUES ('Sabun Cuci', 0);");
    $db->exec("INSERT INTO daftarbelanja VALUES ('Sabun Mandi', 0);");
  } else {
    $db = new SQLite3('daftarbelanja.sqlite');
  }
  return $db;
}
```

❶ Di sini kita menginisialisasi dua variabel, yaitu "$status_code" dan "$json_result", yang secara default berisi *response* tidak ditemukan. Nilai ini akan dikembalikan, jika nantinya tidak ada kondisi yang cocok di program kita.

❷ Selanjutnya, kita cek apakah *request* dikirimkan dengan metode `GET`. Jika ya, ada dua kemungkinan *request*, yaitu mendapatkan detail barang atau mendapatkan daftar barang.

❸ Jika ada informasi setelah "/barang/" (misalnya, "/barang/Tomat"), artinya *request* meminta detail dari barang tertentu (dalam kasus ini Tomat). Jika demikian, kita kembalikan detail dari barang tersebut.

❹ Jika tidak ada informasi nama barang, berarti *request* dikirimkan untuk mendapatkan daftar seluruh barang yang ada.

❺ Jika *request* dikirimkan dengan metode `POST` dan ada parameter "mode", maka ada tiga kemungkinan perintah: menambahkan, membalik status cek, atau menghapus barang.

❻ Jika parameter "mode" berisi "add", kita lakukan operasi menambahkan barang di sini.

❼ Jika parameter "mode" berisi "delete", kita lakukan operasi menghapus barang di sini.

❽ Jika parameter "mode" berisi "toggle", kita lakukan operasi membalik status cek barang di sini.

Jika dijalankan pada *server Apache*, perintah untuk mendapatkan detail, menghapus, dan mengubah status cek barang belanjaan bisa diakses melalui URI http://localhost/barang/index.php/nama-barang. Supaya bisa diakses juga melalui URL http://localhost/barang/nama-barang seperti yang kita inginkan, kita perlu membuat file `.htaccess` pada direktori `barang/` yang isinya seperti berikut:

/barang/.htaccess.

```
RewriteEngine On
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule ^(.*)$ index.php/$1 [L]
```

File tersebut akan mengaktifkan *plug-in rewrite Apache*, yang akan memberikan prefiks `index.php/` pada *path* setelah `barang/`, kecuali jika ditemukan *file* atau *direktori* pada *path* tersebut.

Dari layanan tersebut, kita dapat membuat program *client* seperti di bawah ini.

/chapter3/DaftarBelanja.java.

```
package chapter3;

import java.io.*;
import java.util.*;
import org.json.*;
import org.jsoup.*;
import org.jsoup.helper.*;

public class DaftarBelanja {

  public static void main(String[] args) throws Exception {
    Scanner sc = new Scanner(System.in);
    BarangAPI api = new BarangAPI();
    boolean quit = false;
    while (!quit) { ❶
      System.out.println("1. Daftar Belanja");
      System.out.println("2. Detail Barang");
      System.out.println("3. Tambah Barang");
      System.out.println("4. Hapus Barang");
      System.out.println("5. Balik Status Cek Barang");
      System.out.println("0. Keluar");
      System.out.print("Masukkan pilihan: ");
      int pilihan = Integer.parseInt(sc.nextLine()); ❷
      if (pilihan == 1) {
        Barang[] daftar = api.getBarangList();
        for (Barang barang : daftar) {
          System.out.println(barang);
        }
      } else if (pilihan == 2) {
        System.out.print("Masukkan nama barang untuk dilihat detailnya:
 ");
        String namaBarang = sc.nextLine().trim();
        Barang barang = api.getBarangDetail(namaBarang);
        System.out.println(barang);
      } else if (pilihan == 3) {
        System.out.print("Masukkan nama barang untuk ditambah: ");
        String namaBarang = sc.nextLine().trim();
        api.addBarang(namaBarang);
      } else if (pilihan == 4) {
        System.out.print("Masukkan nama barang untuk dihapus: ");
        String namaBarang = sc.nextLine().trim();
        api.removeBarang(namaBarang);
      } else if (pilihan == 5) {
```

```
        System.out.print("Masukkan nama barang untuk diubah statusnya:
 ");
        String namaBarang = sc.nextLine().trim();
        Barang barang = api.toggleBarang(namaBarang);
        System.out.println(barang);
      } else if (pilihan == 0) {
        quit = true;
      }
    }
    sc.close();
  }
}

class BarangAPI { ❸

  public static final String URL = "http://localhost/barang/";

  public Barang[] getBarangList() throws Exception {
    Connection connection = HttpConnection.connect(URL);
    connection.ignoreContentType(true);
    String content = connection.execute().body();
    JSONObject jsonObject = new JSONObject(content);
    JSONArray items = jsonObject.getJSONArray("items");
    Barang[] barangArray = new Barang[items.length()];
    for (int i = 0; i < items.length(); i++) {
      JSONObject item = items.getJSONObject(i);
      barangArray[i] = new Barang(item.getString("nama"),
 item.getBoolean("cek"));
    }
    return barangArray;
  }

  public Barang getBarangDetail(String namaBarang) throws JSONException,
 IOException {
    Connection connection = HttpConnection.connect(URL + namaBarang);
    connection.ignoreContentType(true);
    String content = connection.execute().body();
    JSONObject jsonObject = new JSONObject(content);
    JSONObject item = jsonObject.getJSONObject("item");
    return new Barang(item.getString("nama"), item.getBoolean("cek"));
  }

  public void addBarang(String namaBarang) throws IOException {
    Connection connection = HttpConnection.connect(URL);
    connection.ignoreContentType(true);
```

```
    connection.method(Connection.Method.POST);
    connection.data("mode", "add");
    connection.data("nama", namaBarang);
    connection.execute();
  }

  public void removeBarang(String namaBarang) throws IOException {
    Connection connection = HttpConnection.connect(URL + namaBarang);
    connection.ignoreContentType(true);
    connection.method(Connection.Method.POST);
    connection.data("mode", "delete");
    connection.data("nama", namaBarang);
    connection.execute();
  }

  public Barang toggleBarang(String namaBarang) throws IOException,
 JSONException {
    Connection connection = HttpConnection.connect(URL + namaBarang);
    connection.ignoreContentType(true);
    connection.method(Connection.Method.POST);
    connection.data("mode", "toggle");
    connection.data("nama", namaBarang);
    String content = connection.execute().body();
    JSONObject jsonObject = new JSONObject(content);
    JSONObject item = jsonObject.getJSONObject("item");
    return new Barang(item.getString("nama"), item.getBoolean("cek"));
  }
}

class Barang { ❹

  public String nama;
  public boolean cek;

  public Barang(String nama, boolean cek) {
    this.nama = nama;
    this.cek = cek;
  }

  @Override
  public String toString() {
    return (this.cek ? "[v]" : "[ ]") + " " + this.nama;
  }

}
```

❶ Aplikasi kali ini menampilkan menu secara berulang-ulang, di mana aplikasi akan menampilkan daftar perintah yang dapat diberikan ke *client*.

❷ Setelah menampilkan daftar perintah, aplikasi menanyakan menu yang ingin dipilih dengan memasukkan angka tertentu. Berdasarkan menu yang dipilih, aplikasi mungkin menanyakan informasi tambahan dan kemudian mengirimkan *request* ke *server* memanfaatkan kelas "BarangAPI" yang kita buat khusus untuk melakukan hal tersebut.

❸ "BarangAPI" merupakan kelas yang bertanggung jawab untuk mengirimkan *request* dan mengurai *response* yang diberikan dari *web service*. Setiap *method* melakukan operasi sesuai namanya, dan memanfaatkan *library* jsoup dan JSON seperti yang digunakan pada bab sebelumnya.

❹ Kelas "Barang" merepresentasikan satu buah barang yang berisi nama dan status cek nya, untuk menyederhanakan parameter dan kembalian dari *method-method* milik _BarangAPI".

3.4. Kesimpulan

Di bab ini kita melihat gaya arsitektur REST dan mengubah *web service* serta *client* Daftar Belanja yang sudah kita buat di bab sebelumnya sehingga mematuhi sebagian dari *constraint* arsitektur tersebut.

3.5. Latihan

1. Saat membahas *constraint* "*Client-Server*", kita mengetahui bahwa sebuah entitas dapat berfungsi sebagai *client* dan *server* sekaligus. Bisakah Anda mengubah kode "barang/index.php" di atas sedemikian sehingga mendapatkan data bukan dari basis data, melainkan dari *web service* yang sudah kita buat di bab sebelumya? ("daftar_json_*.php"). Anda bisa menggunakan fungsi `file_get_contents()` [1] serta `json_decode()` [2] untuk mengirimkan *request* serta mengurai hasil JSON-nya.
2. Cobalah untuk mengimplementasikan *status code* 400 (*Bad Request*) untuk kode "/barang/index.php". Kembalikan status ini jika *client* mengirimkan perintah "add", "delete", atau "toggle", tetapi tidak menyediakan parameter nama.

[1] http://php.net/manual/en/function.file-get-contents.php
[2] http://php.net/manual/en/function.json-decode.php

3. Perintah "add" harusnya menggunakan metode HTTP PUT, dan perintah "delete" harusnya menggunakan metode HTTP DELETE. Cobalah untuk mengubah kode "/barang/index.php" dan "/chapter3/DaftarBelanja.java" di atas sehingga memanfaatkan kedua metode HTTP tersebut alih-alih POST.

4

Web Service Pihak Ketiga

Pada saat buku ini ditulis, terdapat banyak layanan *web service* berbasis REST yang tersedia di internet, baik yang gratis maupun berbayar. Biasanya, layanan diberikan secara gratis namun jumlah *request* dibatasi. Untungnya, sebagian besar *web service* gratis yang tersedia memiliki batas jumlah *request* gratis yang cukup besar jika kita ingin gunakan untuk mencoba-coba.

Di buku ini, kita akan membahas dua buah *web service* yang tersedia secara gratis. Seperti hal-hal lain di web, *web service* yang tersedia tersebut bisa saja sudah berubah saat Anda mencobanya. Silahkan mengacu ke dokumentasi resminya, untuk *update* terakhir dari spesifikasi *web service* yang digunakan.

4.1. Google Places API Web Service [1]

Google Places API Web Service adalah layanan yang disediakan Google untuk mengakses basis data tempat yang dimiliki oleh Google. Google Places API Web Service ini terbagi menjadi 6 layanan:

- **Nearby Search (Place Search)** untuk mendapatkan daftar tempat berdasarkan *query* tertentu.
- **Place Details** untuk mendapatkan informasi detail dari sebuah tempat, misalnya *review* pengguna.
- **Place Add**: untuk menambahkan tempat baru ke basis data tempat milik Google.

[1] https://developers.google.com/places/web-service/

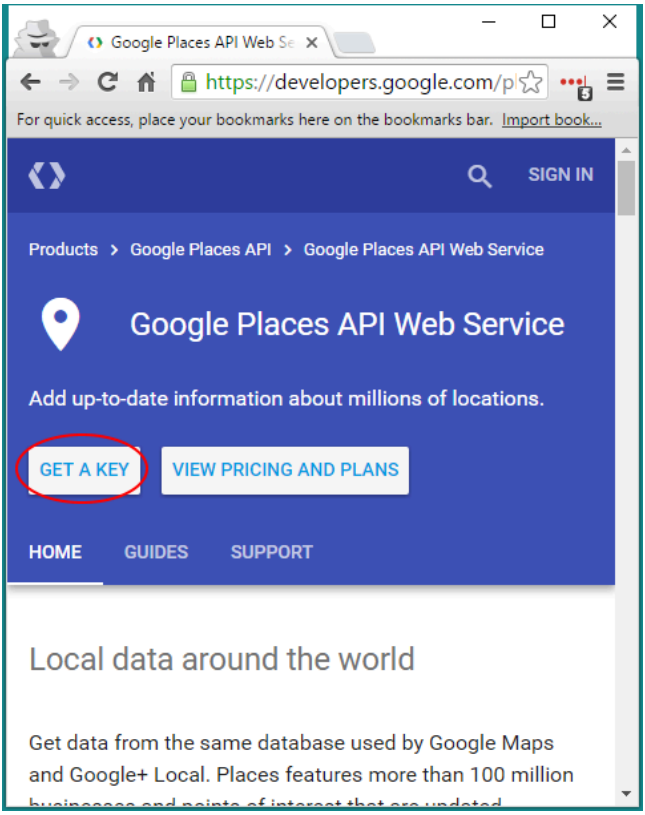
- **Place Photos** untuk mendapatkan foto-foto dari tempat pada basis data Google.
- **Place Autocomplete** untuk membantu fitur *autocomplete* pada antarmuka aplikasi (misal: mengetikkan "Tam" di area Bandung memberikan saran "Taman Film", "Taman Lansia").
- **Query Autocomplete** Mirip Place Autocomplete tetapi mampu mengenali bahasa alami manusia.

Google Places API Web Service mendukung *response* dalam format XML maupun JSON. Buku ini hanya akan membahas format JSON, sedangkan untuk format XML anda dapat mengacu ke dokumentasi resmi.

Saat buku ini ditulis, Google Places API Web Service dapat digunakan secara gratis dengan batasan 1.000 *request* per 24 jam, tetapi jika Anda memverifikasi diri menggunakan kartu kredit, batasannya dinaikkan menjadi 150.000 *request* per 24 jam.

4.1.1. API Key

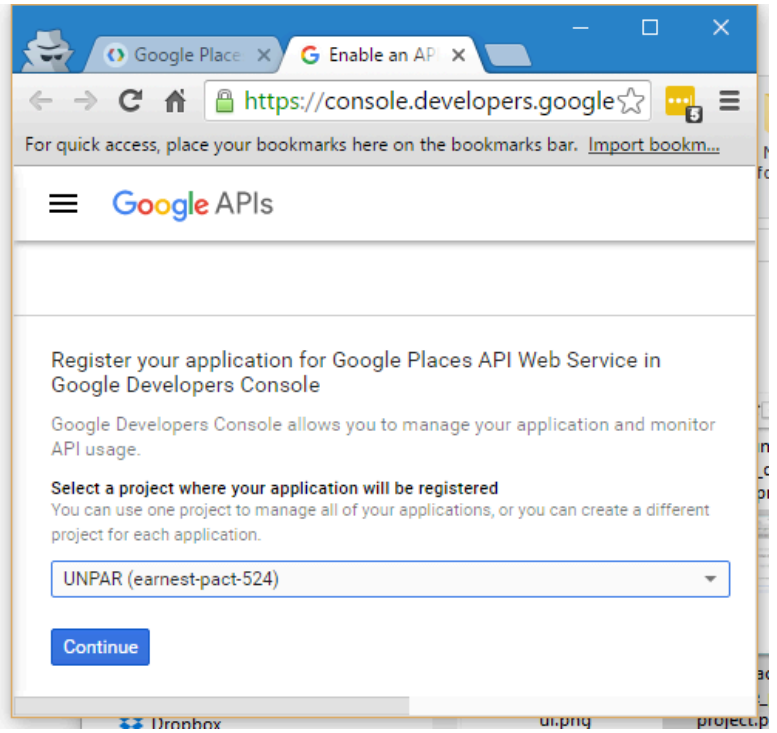
Sama seperti layanan berbasis awan Google lainnya, untuk menggunakan Google Places API Web Service, Anda harus memiliki sebuah *API key*. *Key* ini merupakan *string* acak yang mengidentifikasikan aplikasi yang Anda buat. Usahakan untuk menyimpan *key* ini secara rahasia, karena salah satunya digunakan untuk menghitung kuota *request* yang Anda kirimkan. Untuk mendapatkan sebuah *key* baru, Anda dapat mengklik tombol "Get a Key" pada URL dokumentasi resmi Google Places API Web Service di atas.



Gambar 4.1. Tombol "Get a Key" pada halaman referensi Google Places API Web Service

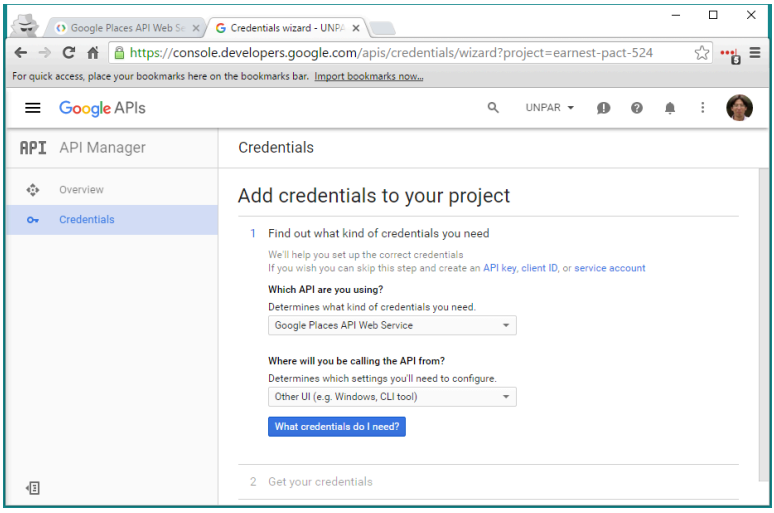
Dari sana, Anda akan dibawa ke situs Google Developers Console [2] untuk mendapatkan *key* yang dimaksud. Jika Anda belum *sign-in*, maka Anda akan diminta untuk *sign-in* dengan akun Google terlebih dahulu. Jika Anda belum membuat *project*, Anda juga akan diminta untuk membuat sebuah *project* untuk menaungi *key* yang akan Anda dapatkan.

[2] https://console.developers.google.com/



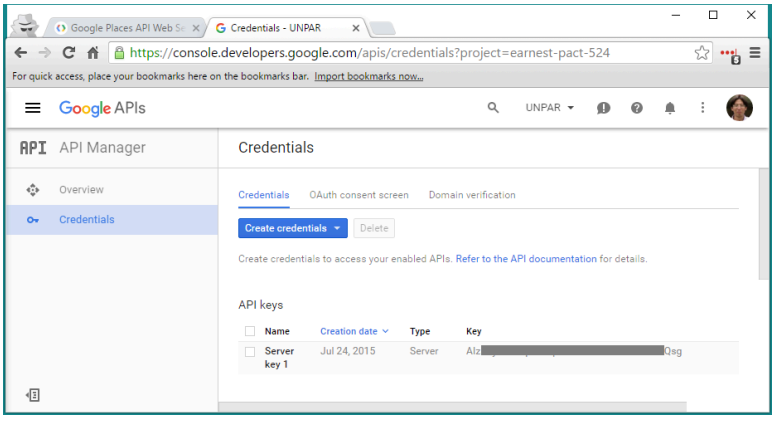
Gambar 4.2. Pilihan project yang menaungi key yang dibuat

Setelah itu, Anda juga akan diminta untuk memilih jenis aplikasi yang Anda buat. Pada kasus ini, kita memilih "Other UI (e.g. Windows, CLI Tool)". Pilihan ini akan memberikan Anda *API key* untuk autentikasi dalam menggunakan layanan Google. Jika Anda memilih jenis aplikasi lain, cara autentikasi mungkin akan berbeda.



Gambar 4.3. Pilihan Jenis Aplikasi yang Dibuat

Pada akhirnya, Anda akan mendapatkan sebuah *key*. Jika Anda sudah pernah membuat *key* yang kompatibel dengan jenis aplikasi yang Anda buat sekarang, Google akan menawarkan untuk menggunakannya kembali. Jika tidak, Google akan membuatkannya untuk Anda.



Gambar 4.4. API Key yang Diberikan Google

4.1.2. Nearby Search API

Nearby Search merupakan salah satu layanan dari Google Places API Web Service, yang digunakan untuk mencari tempat-tempat menarik pada lokasi

tertentu. Misalnya, untuk mencari "Taman Film" yang ada di kota Bandung, atau mencari rumah makan yang berada di dekat Monumen Nasional, Jakarta.

Seperti didefinisikan pada dokumentasinya [3], untuk menggunakan layanan *Nearby Search* dalam format JSON, kita perlu mengirimkan *GET request* ke URL berikut:

```
https://maps.googleapis.com/maps/api/place/nearbysearch/json
```

Layanan ini memerlukan beberapa parameter yang dikirimkan memanfaatkan parameter *GET*. Parameter-parameter yang wajib antara lain:

- `key` berisi API key yang telah kita dapatkan sebelumnya, misalnya "Aiz…Qsg".
- `location` berisi lokasi pusat pencarian, dalam format *latitude*,*longitude*.
- `radius` berisi jarak maksimal (dalam satuan meter) hasil pencarian, dihitung dari pusat pencarian.
- Salah satu dari `keyword`, `name`, atau `type`:
  - `keyword` berisi kata kunci yang digunakan dalam pencarian. *Keyword* ini akan dicocokkan dengan nama tempat, jenis tempat, alamat, dll.
  - `name` berisi kata kunci yang digunakan dalam pencarian. *Name* hanya akan dicocokkan dengan nama tempat saja.
  - `type` berisi kode jenis tempat yang ingin dicari, seperti `restaurant`, `park`, dll. Daftar lengkap dari kode jenis tempat yang didukung dapat dilihat di dokumentasi resminya [4].

> *Latitude* (lintang) dan *longitude* (bujur) adalah dua buah bilangan desimal yang dapat digunakan untuk merepresentasikan sebuah lokasi di bumi. Titik 0,0 merepresentasikan posisi lintang 0 dan bujur 0, sedangkan kota Bandung berada pada koordinat -6.916667,107.6 (sumber: Wikipedia).

[3] https://developers.google.com/places/web-service/search
[4] https://developers.google.com/places/supported_types

Sebagai contoh, untuk menemukan Taman Film di kota Bandung, kita dapat mengirimkan *GET request* ke URL berikut(ganti API key dengan milik Anda):

```
https://maps.googleapis.com/maps/api/place/nearbysearch/json?
key=AIz...Qsg&location=-6.916667,107.6&radius=10000&name=Taman+Film
```

Saat buku ini ditulis, perintah tersebut memberikan *response* seperti berikut:


```
{
   "html_attributions" : [],
   "results" : [
      {
         "geometry" : {
            "location" : {
               "lat" : -6.898574,
               "lng" : 107.607645
            }
         },
         "icon" : "https://maps.gstatic.com/mapfiles/place_api/icons/
generic_recreational-71.png",
         "id" : "4515ad9087f8fba947bf22051500eb5f3dda7fda3",
         "name" : "Taman Film",
         "opening_hours" : {
            "open_now" : false,
            "weekday_text" : []
         },
         "photos" : [
            {
               "height" : 3672,
               "html_attributions" : [
                  "\u003ca href=\"https://maps.google.com/maps/
contrib/115271990533873034146/photos\"\u003evranxzs tribals\u003c/a
\u003e"
               ],
               "photo_reference"
 : "CoQBcwAAAJBruGOOOAXEI1KbMD5uAug73ic401f9uojrTFF4WNgmPqyQqgAEm...",
               "width" : 4896
            }
         ],
         "place_id" : "ChIJqycGPUXmaC4RMexoONkWNEA",
         "rating" : 4.2,
         "reference"
 : "CmRdAAAAeYYbUMsi9SOcb4A1CcSJwKaOVTVqsBtt8H2ZUpF_r61ygU2ySEbr28g...",
         "scope" : "GOOGLE",
```

```
         "types" : [ "park", "point_of_interest", "establishment" ],
         "vicinity" : "Bandung, Jl. Layang Pasupati, Bandung Wetan"
      }
   ],
   "status" : "OK"
}
```

Hasil yang Anda dapatkan belum tentu sama dengan hasil di atas, karena sifat dari *web service* yang dinamis, dan dapat berubah sesuai keadaan. Walaupun begitu, format *response* biasanya tidak berubah.

Berikut adalah penjelasan dari setiap anggota JSON yang diberikan:

Tabel 4.1. Deskripsi Response Nearby Search API

Anggota	Deksripsi
`status`	Status *response* dari *request* yang dikirimkan, dapat berupa: • `OK` jika *request* berhasil, dan *response* akan berisi hasil pencarian. • `ZERO_RESULTS` jika tidak ada tempat yang dituemukan. • `OVER_QUERY_LIMIT` jika jumlah *request* sudah melebihi kuota. • `REQUEST_DENIED` jika permintaan ditolak (biasanya karena tidak ada `key` pada *request*). • `INVALID_REQUEST` jika ada parameter wajib yang tidak diberikan, atau ada parameter yang tidak valid.
`html_attributions`	*Array* yang berisi teks-teks atribusi yang harus ditampilkan kepada pengguna.
`results`	*Array* dari objek JSON yang berisi informasi detail dari setiap tempat yang ditemukan. Hasilnya dibatasi 20 tempat saja, dan untuk mendapatkan sisanya dapat memanfaatkan nilai `next_page_token`. Elemen dari `results` dijelaskan pada tabel berikut.

Anggota	Deksripsi
`next_page_token`	Nilai dari atribut ini dapat digunakan sebagai parameter `pagetoken` pada *request* berikutnya, untuk mendapatkan 20 tempat berikutnya.

Setiap elemen dari `results` adalah objek dengan spesifikasi seperti berikut:

Tabel 4.2. Deskripsi Response Nearby Search API, elemen `results`

Anggota	Deskripsi
`icon`	URL dari gambar simbol yang merepresentasikan jenis tempat tersebut.
`geometry`	Informasi geometri dari tempat ini, biasanya berupa objek yang mengandung anggota `location`, yang berisi dua anggota `lat` dan `lng` yang menyatakan *latitude* dan *longitude* dari tempat tersebut.
`name`	Berisi nama dari tempat tersebut.
`opening_hours`	Objek JSON yang mengandung satu elemen `open_now` berupa *boolean* yang menyatakan apakah tempat tersebut buka pada saat *request* dikirimkan.
`photos`	*Array* yang berisi maksimal satu elemen berupa objek JSON yang memberikan informasi foto dari tempat tersebut. Anggota dari objek JSON ini dijelaskan pada tabel di bawah.
`place_id`	*String* pengenal dari tempat ini.
`scope`	Berisi salah satu dari `GOOGLE` atau `APP`. `GOOGLE` berarti tempat tersebut publik dan muncul di seluruh hasil pencarian. `APP` berarti tempat tersebut hanya muncul pada hasil pencarian dengan *api key* yang sedang digunakan.
`alt_ids`	*Array* yang elemennya berisi objek JSON, yang mengandung dua anggota: `place_id`

Anggota	Deskripsi
	yang menandakan alternatif `place_id` dari tempat tersebut, dan `scope` yang selalu berisi `APP`.
`price_level`	Tingkat harga dari tempat tersebut, berupa bilangan bulat dari `0` yang berarti gratis sampai `4` yang berarti sangat mahal. Tidak semua tempat memiliki anggota ini, dan harga riil nya berbeda-beda di setiap daerah.
`rating`	Berisi nilai *rating* rata-rata yang diberikan pengguna, berupa bilangan desimal antara 1.0 sampai 5.0.
`types`	*Array* dari kode tipe tempat ini. Daftar kode tipe dapat dilihat di dokumentasi resminya [a].
`vicinity`	Alamat dari tempat ini.
`permanently_closed`	*Boolean* yang berisi `true` jika tempat ini sudah tutup secara permanen. Jika belum, anggota ini tidak ada.

[a] https://developers.google.com/places/supported_types

Anggota-anggota dari elemen `photos` dijelaskan pada tabel berikut.

Tabel 4.3. Deskripsi Response Nearby Search API, elemen `photos`

Anggota	Deskripsi
`photo_reference`	Sebuah *string* yang dapat digunakan untuk mendapatkan *file* foto tersebut, melalui Place Photos API. API ini tidak dibahas pada buku ini, tetapi dapat dipelajari pada dokumentasi resminya [a].
`height`	Lebar maksimum yang dapat diminta dari foto tersebut dalam satuan piksel.
`width`	Panjang maksimum yang dapat diminta dari foto tersebut dalam satuan piksel.

Anggota	Deskripsi
`html_attributions`	*Array* dari *string* yang menyatakan atribusi yang harus ditampilkan kepada pengguna.

[a] https://developers.google.com/places/web-service/photos

Tidak semua atribut pasti muncul dalam hasil pencarian. Program yang Anda buat sebaiknya menangani hal tersebut.

Berikut adalah program sederhana yang menanyakan ke pengguna sebuah kata kunci, dan mencari tempat di Bandung berdasarkan kata kunci tersebut:

```
import java.util.*;
import org.json.*;
import org.jsoup.*;
import org.jsoup.helper.*;

public class PlaceSearchDemo {

  public static final String PLACESEARCH_URL = "https://
maps.googleapis.com/maps/api/place/nearbysearch/json";
  public static final String APIKEY = "AIz...Qsg"; /* TODO isi dengan
 API key Anda */
  public static final String CITY_LOCATION = "-6.916667,107.6";
  public static final int CITY_RADIUS = 10000;

  public static void main(String[] args) throws Exception {
    Scanner sc = new Scanner(System.in);
    System.out.print("Cari apa di Bandung? ");
    String query = sc.nextLine();
    Connection connection = HttpConnection.connect(PLACESEARCH_URL);
    connection.data("key", APIKEY);
    connection.data("location", CITY_LOCATION);
    connection.data("radius", "" + CITY_RADIUS);
    connection.data("name", query);
    connection.ignoreContentType(true);
    String content = connection.execute().body();
    JSONObject jsonResponse = new JSONObject(content);
    if (jsonResponse.getString("status").equals("OK")) {
      JSONArray results = jsonResponse.getJSONArray("results");
      System.out.println("Hasil:");
      System.out.println("---");
      for (int i = 0; i < results.length(); i++) {
```

```
        JSONObject result = results.getJSONObject(i);
        System.out.println("Nama: " + result.optString("name", "-"));
        System.out.println("Alamat: " +
 result.optString("vicinity", "-"));
        System.out.println("Rating: " + result.optDouble("rating",
 Double.NaN));
        System.out.println("---");
      }
    } else {
      System.err.println(jsonResponse.getString("status"));
    }

    sc.close();
  }
}
```

4.2. Google Maps Directions API [5]

Google Maps Directions API adalah layanan lain dari Google yang memungkinkan Anda mendapatkan rute terdekat dari satu lokasi ke lokasi lain.

Untuk mendapatkan rute terdekat tersebut dalam format JSON, kita perlu mengirimkan *request* ke URL:

```
https://maps.googleapis.com/maps/api/directions/json
```

Layanan ini memerlukan beberapa parameter yang dikirimkan memanfaatkan parameter *GET*. Parameter-parameter yang wajib antara lain:

- `key` berisi API key yang telah kita dapatkan sebelumnya, misalnya `Aiz...Qsg`.
- `origin` berisi lokasi asal pencarian rute, bisa dalam salah satu dari:
  - Lokasi dalam format *latitude,longitude*,
  - Teks alamat, di mana Google akan memperkirakan lokasinya, atau
  - *Place ID* seperti yang didapat dari Google Place Search API di atas. Place ID yang diberikan didahului oleh kata kunci `place_id:`
- `destination` berisi lokasi akhir pencarian rute. Sama dengan `origin`, bisa berisi lokasi dalam *latitude,longitude*, teks alamat, maupun Place ID.

[5] https://developers.google.com/maps/documentation/directions

Ada juga parameter-parameter opsional yang dapat diberikan, antara lain:

- `mode` menyatakan moda perjalanan yang diinginkan, bisa berupa `driving` (menyetir), `walking` (berjalan kaki), `bicycling` (bersepeda), atau `transit` (menggunakan transportasi publik). Jika tidak diberikan, `driving` akan digunakan.
- `waypoints` menyatakan satu atau lebih lokasi yang dipisahkan dengan simbol pipa `|`. Setiap lokasi bisa direpresentasikan dalam format *latitude,longitude*, teks alamat, maupun *place id* (sama seperti `origin` dan `destination`. Lokasi pertama juga bisa diisi dengan `optimize:true`, yang menyatakan bahwa Google dapat mengubah urutan *waypoint* jika urutan yang baru menghasilkan rute yang lebih baik.
- `alternatives` menyatakan apakah Google perlu mencari alternatif rute (diisi `true`) atau cukup satu saja yang terbaik (diisi `false`).
- `avoid` menyatakan hal-hal yang harus dihindari pada hasil rute. Bisa berisi `tolls` (jalan tol), `highways` (jalan besar), `ferries` (kapal feri), atau `indoor` (melewati gedung, khusus jika `mode` berisi `walking` atau `transit`). Bisa juga kombinasi dari keempatnya, dengan dipisahkan simbol pipa “|”.
- `language` menyatakan bahasa yang akan digunakan pada hasil pencarian. Untuk Bahasa Indonesia, nilainya dapat diisi dengan `id`.
- `units` menyatakan satuan yang dipakai, berupa `metric` (kilometer dan meter) atau `imperial` (mil dan kaki).
- `departure_time` menyatakan waktu keberangkatan, yang dapat diisi dengan waktu di masa depan. Google akan memprediksi waktu perjalanan dengan memperhitungkan faktor kemacetan lalu lintas berdasarkan data terdahulu. Nilainya diisi dalam format *UNIX Time*, yaitu jumlah detik sejak 1 Januari 1970 tengah malam (GMT).
- `arrival_time` menyatakan waktu sampai yang diinginkan. Hanya salah satu dari `departure_time` atau `arrival_time` dapat dikirimkan pada *request*.
- `traffic_model` menyatakan asumsi yang digunakan dalam menghitung `duration_in_traffic` pada hasil, yaitu perkiraan waktu perjalanan dengan memperhitungkan kemacetan. Nilainya bisa berisi `best_guess` (seakurat mungkin), `pessimistic` (pesimis), atau `optimistic` (optimis).

- `transit_mode` menyatakan moda transportasi publik yang ingin digunakan, khusus jika `mode` berisi `transit`. Nilainya bisa berupa `bus`, `subway`, `train`, `tram`, atau kombinasi keempatnya, dipisahkan dengan simbol pipa `|`. Bisa juga berisi `rail` yang ekivalen dengan `train|tram|subway`.
- `transit_routing_preference` menyatakan preferensi dalam menggunakan transportasi publik, khusus jika `mode` berisi `transit`. Nilainya bisa berupa `less_walking` (sedikit berjalan) atau `fewer_transfers` (sedikit berganti moda transportasi).

Sebagai contoh, untuk menemukan rute menyetir dari Taman Film ke Taman Lansia di Bandung dalam Bahasa Indonesia, kita dapat mengirimkan perintah seperti berikut:

```
https://maps.googleapis.com/maps/api/directions/json?
    origin=-6.898574,107.607645
    &destination=-6.9110685,107.6134872
    &language=id
```

Saat buku ini ditulis, perintah tersebut memberikan *response* seperti berikut:


```
{
   "geocoded_waypoints" : [
      {
         "geocoder_status" : "OK",
         "place_id" : "ChIJof6vPUXmaC4Ro_OSUmlPfvc",
         "types" : [ "route" ]
      },
      {
         "geocoder_status" : "OK",
         "place_id" : "ChIJM6yvlDbmaC4R861aOlGV1Co",
         "types" : [ "route" ]
      }
   ],
   "routes" : [
      {
         "bounds" : {
            "northeast" : {
               "lat" : -6.8986138,
               "lng" : 107.6140051
            },
            "southwest" : {
```

```
               "lat" : -6.910949899999999,
               "lng" : 107.6042508
            }
         },
         "copyrights" : "Data peta ©2016 Google",
         "legs" : [
            {
               "distance" : {
                  "text" : "2,5 km",
                  "value" : 2475
               },
               "duration" : {
                  "text" : "7 menit",
                  "value" : 430
               },
               "end_address" : "Jl. Kalimantan, Merdeka, Sumur Bandung,
Kota Bandung, Jawa Barat, Indonesia",
               "end_location" : {
                  "lat" : -6.910949899999999,
                  "lng" : 107.6140051
               },
               "start_address" : "Jl. Layang Pasupati, Taman Sari,
Bandung Wetan, Kota Bandung, Jawa Barat, Indonesia",
               "start_location" : {
                  "lat" : -6.8986138,
                  "lng" : 107.6076683
               },
               "steps" : [
                  {
                     "distance" : {
                        "text" : "0,2 km",
                        "value" : 206
                     },
                     "duration" : {
                        "text" : "1 menit",
                        "value" : 18
                     },
                     "end_location" : {
                        "lat" : -6.899496099999999,
                        "lng" : 107.6060237
                     },
                     "html_instructions" : "Ke arah \u003cb\u003ebarat
daya\u003c/b\u003e di \u003cb\u003eJl. Layang Pasupati\u003c/b\u003e",
                     "polyline" : {
                        "points" : "hkbi@}bxoSFJRf@P^HPr@bBdA~B"
                     },
```

```
                  "start_location" : {
                     "lat" : -6.8986138,
                     "lng" : 107.6076683
                  },
                  "travel_mode" : "DRIVING"
               },
               {
                  "distance" : {
                     "text" : "0,2 km",
                     "value" : 188
                  },
                  "duration" : {
                     "text" : "1 menit",
                     "value" : 33
                  },
                  "end_location" : {
                     "lat" : -6.900303099999999,
                     "lng" : 107.604542
                  },
                  "html_instructions" : "Ambil jalan keluar menuju
 \u003cb\u003eJl. Cihampelas\u003c/b\u003e",
                  "maneuver" : "ramp-left",
                  "polyline" : {
                     "points" : "zpbi@sxwoSNNFHx@dBZp@Vr@X`A"
                  },
                  "start_location" : {
                     "lat" : -6.899496099999999,
                     "lng" : 107.6060237
                  },
                  "travel_mode" : "DRIVING"
               },
               {
                  "distance" : {
                     "text" : "0,5 km",
                     "value" : 455
                  },
                  "duration" : {
                     "text" : "1 menit",
                     "value" : 76
                  },
                  "end_location" : {
                     "lat" : -6.9043106,
                     "lng" : 107.6043441
                  },
                  "html_instructions" : "Tetap di \u003cb\u003ekiri
\u003c/b\u003e di pertigaan dan bergabung ke \u003cb\u003eJl. Cihampelas
```

```
\u003c/b\u003e\u003cdiv style=\"font-size:0.9em\"\u003eLewati Indomaret
 (di kiri)\u003c/div\u003e",
                     "maneuver" : "fork-left",
                     "polyline" : {
                        "points"
 : "zubi@kowoSZVLHLFRDp@HtBGn@AfBAnBAbGA`@A"
                     },
                     "start_location" : {
                        "lat" : -6.900303099999999,
                        "lng" : 107.604542
                     },
                     "travel_mode" : "DRIVING"
                  },
                  {
                     "distance" : {
                        "text" : "79 m",
                        "value" : 79
                     },
                     "duration" : {
                        "text" : "1 menit",
                        "value" : 12
                     },
                     "end_location" : {
                        "lat" : -6.9042604,
                        "lng" : 107.6049942
                     },
                     "html_instructions" : "Di bundaran, ambil jalan
 keluar \u003cb\u003epertama\u003c/b\u003e menuju \u003cb\u003eJl.
 Wastukancana\u003c/b\u003e",
                     "maneuver" : "roundabout-left",
                     "polyline" : {
                        "points" : "|nci@cnwoS?A?A?A@??A?A@??A?A@??A@?
@??A@?@?@?Ec@Ga@AAIg@"
                     },
                     "start_location" : {
                        "lat" : -6.9043106,
                        "lng" : 107.6043441
                     },
                     "travel_mode" : "DRIVING"
                  },
                  {
                     "distance" : {
                        "text" : "0,5 km",
                        "value" : 493
                     },
                     "duration" : {
```

```
                        "text" : "1 menit",
                        "value" : 81
                     },
                     "end_location" : {
                        "lat" : -6.906517699999999,
                        "lng" : 107.6079207
                     },
                     "html_instructions" : "Belok sedikit ke \u003cb
\u003ekiri\u003c/b\u003e setelah Sekolah Tinggi Desain Indonesia (di
 sebelah kiri)\u003cdiv style=\"font-size:0.9em\"\u003eLewati Hotel
 California Bandung (di kiri)\u003c/div\u003e",
                     "maneuver" : "turn-slight-left",
                     "polyline" : {
                        "points"
 : "rnci@erwoS[]EIGSCMGYGa@Dc@Hc@Nm@J_@Ti@LUT]PSFGb@c@LKPO..."
                     },
                     "start_location" : {
                        "lat" : -6.9042604,
                        "lng" : 107.6049942
                     },
                     "travel_mode" : "DRIVING"
                  },
                  {
                     "distance" : {
                        "text" : "0,3 km",
                        "value" : 286
                     },
                     "duration" : {
                        "text" : "1 menit",
                        "value" : 64
                     },
                     "end_location" : {
                        "lat" : -6.907016899999999,
                        "lng" : 107.6104576
                     },
                     "html_instructions" : "\u003cb\u003eJl.
 Wastukancana\u003c/b\u003e belok \u003cb\u003ekiri\u003c/b\u003e dan
 menjadi \u003cb\u003eJl. L. L. RE.Martadinata\u003c/b\u003e\u003cdiv
 style=\"font-size:0.9em\"\u003eLewati Honda (di kanan)\u003c/div
\u003e",
                     "polyline" : {
                        "points" : "v|ci@odxoSPuA@KP_BNyA@IHu@`@_D"
                     },
                     "start_location" : {
                        "lat" : -6.906517699999999,
                        "lng" : 107.6079207
```

```
                  },
                  "travel_mode" : "DRIVING"
               },
               {
                  "distance" : {
                     "text" : "0,3 km",
                     "value" : 330
                  },
                  "duration" : {
                     "text" : "1 menit",
                     "value" : 78
                  },
                  "end_location" : {
                     "lat" : -6.9098479,
                     "lng" : 107.6106187
                  },
                  "html_instructions" : "Belok \u003cb\u003ekanan
\u003c/b\u003e setelah Berlian Merdeka (di sebelah kanan)\u003cdiv
 style=\"font-size:0.9em\"\u003eLewati PT. Asaba Cab. Bdg (di
 kiri)\u003c/div\u003e",
                  "maneuver" : "turn-right",
                  "polyline" : {
                     "points" : "z_di@ktxoS?YXLzCIzACd@?pCAjBC"
                  },
                  "start_location" : {
                     "lat" : -6.907016899999999,
                     "lng" : 107.6104576
                  },
                  "travel_mode" : "DRIVING"
               },
               {
                  "distance" : {
                     "text" : "0,4 km",
                     "value" : 382
                  },
                  "duration" : {
                     "text" : "1 menit",
                     "value" : 60
                  },
                  "end_location" : {
                     "lat" : -6.910462799999999,
                     "lng" : 107.6138765
                  },
                  "html_instructions" : "Belok \u003cb\u003ekiri
\u003c/b\u003e setelah Yogya Merdeka (di sebelah kiri)\u003cdiv style=
\"font-size:0.9em\"\u003eLewati South Bank (di kiri)\u003c/div\u003e",
```

```
                     "maneuver" : "turn-left",
                     "polyline" : {
                        "points" : "pqdi@kuxoSA]?S?QDaCEsA?K?
IKMGSFa@Le@DOh@uAp@cBN[@CBC@C"
                     },
                     "start_location" : {
                        "lat" : -6.9098479,
                        "lng" : 107.6106187
                     },
                     "travel_mode" : "DRIVING"
                  },
                  {
                     "distance" : {
                        "text" : "56 m",
                        "value" : 56
                     },
                     "duration" : {
                        "text" : "1 menit",
                        "value" : 8
                     },
                     "end_location" : {
                        "lat" : -6.910949899999999,
                        "lng" : 107.6140051
                     },
                     "html_instructions" : "Terus lurus ke \u003cb
\u003eJl. Kalimantan\u003c/b\u003e\u003cdiv style=\"font-size:0.9em
\"\u003eTujuan ada di sebelah kanan.\u003c/div\u003e",
                     "maneuver" : "straight",
                     "polyline" : {
                        "points" : "judi@wiyoSPEHCHAHCL?b@I"
                     },
                     "start_location" : {
                        "lat" : -6.910462799999999,
                        "lng" : 107.6138765
                     },
                     "travel_mode" : "DRIVING"
                  }
               ],
               "via_waypoint" : []
            }
         ],
         "overview_polyline" : {
            "points"
 : "hkbi@}bxoSv@dBxBbFVXtAvCp@tBh@`@`@Lp@HtBGvCCrJC`@C?C@CBC@ADA@?
MeAKi@a@g@Ka@O{@NgAZmAb@_Af@q@j@k@^[f@YtBm@nBSn@WPuARkBPcBj@uE?
YXLvFMvDAjBCA]?e@DaCEsA?UKMGSFa@Le@n@eB`A_CDGRIj@Ib@I"
```

```
            },
            "summary" : "Jl. Wastukancana",
            "warnings" : [],
            "waypoint_order" : []
         }
      ],
      "status" : "OK"
}
```

Berikut adalah penjelasan dari setiap anggota JSON yang diberikan:

Tabel 4.4. Deskripsi Response Directions API

Anggota	Deksripsi
`status`	Status *response* dari *request* yang dikirimkan, isinya dapat berupa salah satu dari: • `OK` jika *request* berhasil, dan *response* akan mengandung informasi tambahan terkait hasil pencarian. • `NOT_FOUND` jika salah satu dari `origin` atau `destination` bukan berupa *latitude,longitude* dan tidak dapat ditemukan. • `ZERO_RESULTS` jika Google tidak berhasil menemukan rute yang diminta. • `INVALID_REQUEST` jika ada parameter wajib yang tidak diberikan, atau ada parameter yang tidak valid. • `OVER_QUERY_LIMIT` yang berarti jumlah *request* sudah melebihi kuota. • `REQUEST_DENIED` jika permintaan ditolak.
`geocoded_waypoints`	Hasil *geocoding* dari `origin`, `destination`, maupun *waypoints* pada *request*. *Geocoding* pada API ini adalah proses konversi dari lokasi maupun nama tempat menjadi `place_id`.

Anggota	Deksripsi
`routes`	*Array* dari objek JSON yang berisi informasi detail dari setiap alternatif rute yang ditemukan. Elemen dari `routes` dijelaskan pada tabel di bawah.

Setiap elemen dari `routes` adalah objek yang memiliki anggota seperti berikut.

Tabel 4.5. Deskripsi Response Directions API, elemen *array* `routes`

Anggota	Deksripsi
`summary`	Ringkasan dari alternatif rute ini, untuk membedakan dengan rute alternatif lainnya.
`legs`	*Array* yang berisi objek JSON yang merepresentasikan *leg*. *Leg* adalah subrute untuk setiap *waypoint* yang diberikan (jika parameter opsional `waypoints` diberikan). Jika `waypoints` tidak diberikan, *array* ini akan berisi satu elemen saja. Penjelasan setiap elemen `legs` dijelaskan pada tabel di bawah.
`waypoint_order`	*Array* yang berisi urutan *waypoint* yang baru, jika parameter `waypoints` diawali dengan `optimized:true`.
`overview_polyline`	Berisi daftar titik-titik yang dilalui oleh rute yang didapatkan. Titik-titik rute ini sudah disederhanakan (tidak detail), dan diringkas dengan format *encoded polyline*. Format ini akan dijelaskan kemudian.
`bounds`	Menyatakan kotak yang menyelubungi rute yang diberikan. Kotak ini direpresentasikan dalam objek JSON, yang mengandung dua anggota `northeast` (kanan-atas) dan `southwest` (kiri-bawah). Setiap anggota berupa objek JSON lain yang mengandung dua anggota `lat` (*latitude*) dan `lng` (*longitude*).

Anggota	Deksripsi
`copyrights`	Berisi teks *copyright* yang harus ditampilkan kepada pengguna.
`warnings`	*Array string* yang berisi peringatan yang harus ditampilkan kepada pengguna, jika ada.
`fare`	Informasi biaya transportasi publik yang harus dikeluarkan, jika parameter `mode` berisi `transit` dan Google memiliki informasi tarif untuk setiap moda yang digunakan. Informasi ini belum tersedia di Indonesia.

Tabel 4.6. Deskripsi Response Directions API, elemen *array* `legs`

Anggota	Deksripsi
`steps`	*Array* yang berisi objek JSON yang menyatakan setiap langkah yang harus diambil. Penjelasan setiap elemen `steps` dijelaskan pada tabel di bawah.
`distance`	Menyatakan jarak yang harus ditempuh pada *leg* ini, berupa objek JSON yang berisi dua anggota `value` (angka yang menyatakan jarak dalam meter) dan `text` (jarak dalam format teks yang dapat dibaca manusia, misalnya "10 meter").
`duration`	Menyatakan waktu yang dibutuhkan untuk menempuh *leg* ini, berupa objek JSON yang berisi dua anggota `value` (angka yang menyatakan waktu dalam detik) dan `text (waktu yang dibutuhkan dalam format teks yang dapat dibaca manusia, misalnya "10 menit").
`duration_in_traffic`	Sama seperti `duration`, tetapi memperhitungkan faktor kepadatan lalu lintas.

Anggota	Deksripsi
`arrival_time` dan `departure_time`	Waktu sampai dan keberangkatan, jika parameter `mode` berisi `transit`. Berupa objek JSON yang mengandung tiga anggota `value` (waktu sampai sesuai dengan objek `Date` pada javascript), `text` (waktu sampai dalam format teks yang dapat dibaca manusia), dan `time_zone` (zona waktu pada lokasi akhir *leg*).
`start_location` dan `end_location`	Berisi lokasi awal dan akhir dari *leg* ini, berupa objek JSON yang memiliki dua anggota `lat` (*latitude*) dan `lng` (*longitude*).
`start_address` dan `end_address`	Berisi lokasi awal dan akhir *leg* ini, dalam format teks yang dapat dibaca manusia.

Tabel 4.7. Deskripsi Response Directions API, elemen *array* `steps`

Anggota	Deksripsi
`html_instructions`	Berisi instruksi *step* ini, dalam format HTML.
`distance`	Jarak dari *step* ini, dengan format yang sama seperti anggota `distance` pada elemen `legs` di atas.
`duration`	Waktu tempuh *step* ini, dengan format yang sama seperti anggota `duration` pada elemen `legs` di atas.
`start_location` dan `end_location`	Lokasi awal dan akhir dari *step* ini, dengan format yang sama seperti anggota `start_location` dan `end_location` pada elemen `legs` di atas.
`polyline`	Berisi daftar titik-titik yang dilalui pada *step* ini. Titik-titik rute ini diringkas dengan format *encoded polyline*. Format ini akan dijelaskan kemudian.
`steps`	*Array* yang berisi *sub-step* dari *step* ini, jika parameter `mode` berisi `transit`. Formatnya sama dengan elemen `step` ini.

Anggota	Deksripsi
`transit_details`	Berisi detail transit, jika parameter `mode` berisi `transit`. Penjelasan objek `transit_details` dijelaskan pada tabel di bawah.

Algoritma *encoded polyline* terlalu rumit untuk dijelaskan di buku ini. Untuk mempelajarinya Anda dapat mengakses dokumentasi resminya [6].

Tabel 4.8. Deskripsi Response Directions API, objek `transit_details`

Anggota	Deksripsi
`arrival_stop` dan `departure_stop`	Berisi informasi halte atau stasiun dari *leg* ini, pada tujuan maupun keberangkatan. Berupa objek JSON yang mengandung dua anggota `name` (nama halte atau stasiun) dan `location` (objek JSON yang mengandung dua anggota `lat` (*latitude*) dan `lng` (*longitude*)).
`arrival_time` dan `departure_time`	Waktu sampai dan berangkat dari *leg* ini, berupa objek JSON dengan tiga anggota `text` (Waktu dalam format yang dapat dibaca manusia), `value` (Waktu dalam format UNIX time, yaitu jumlah detik sejak 1 Januari 1970 GMT), dan `time_zone` (kode zona waktu yang digunakan di halte atau stasiun ini).
`headsign`	Arah yang harus diambil saat naik dari halte atau stasiun ini. Biasanya berisi nama terminal akhir.
`headway`	Interval keberangkatan di halte / stasiun ini, dalam detik.
`num_stops`	Jumlah halte yang harus dilewati sebelum turun. Halte atau stasiun untuk turun dihitung, tetapi halte atau stasiun keberangkatan tidak dihitung.

[6] https://developers.google.com/maps/documentation/utilities/polylinealgorithm

Anggota	Deksripsi
`line`	Berisi informasi mengenai jalur yang harus diambil pada *leg* ini, dalam bentuk objek JSON. Penjelasan objek `line` dijelaskan pada tabel di bawah.

Tabel 4.9. Deskripsi Response Directions API, objek `line`

Anggota	Deksripsi
`name`	Berisi nama jalur ini.
`short_name`	Berisi nama jalur yang lebih singkat, biasanya kode jalur.
`color`	Warna yang umum digunakan untuk merepresentasikan jalur ini, dalam format string heksadesimal (misal, #FF0000).
`agencies`	*Array* yang tiap elemennya berupa objek JSON yang merepresentasikan penyedia layanan, dan mengandung tiga anggota yaitu `name` (nama penyedia layanan), `url` (alamat situs web), `phone` nomer telepon. Informasi ini wajib ditampilkan ke pengguna.
`url`	Alamat situs web dari jalur ini.
`icon`	URL untuk mendapatkan gambar yang merepresentasikan jalur ini.
`text_color`	Berisi warna yang umum digunakan untuk teks yang merepresentasikan jalur ini, dalam format string heksadesimal (misal, #FF0000).
`vehicle`	Berisi informasi kendaraan yang digunakan pada jalur ini, dalam bentuk objek JSON yang mengandung empat anggota `name` (nama kendaraan), `type` (tipe kendaraan), `icon` (URL gambar yang merepresentasikan kendaraan), `local_icon` (URL gambar yang merepresentasikan kendaraan secara lokal).

4.3. Web Service Pihak Ketiga Lainnya

Ada banyak *web service* di luar sana yang dapat Anda manfaatkan, terutama dari perusahaan-perusahaan besar. Microsoft memiliki sejumlah *web service* yang tergabung dalam Cognitive Services [7], sebagian besar berbasis pembelajaran mesin. Yahoo memiliki sejumlah *web service* [8] yang sebagian besar digunakan untuk mengakses layanan yang mereka miliki seperti Weather atau Flickr. Di Indonesia, Tiket.com menyediakan sejumlah *web service* [9] untuk pembelian tiket pesawat, hotel, dll. Selain itu, situs ProgrammableWeb memiliki direktori *web service* yang dapat digunakan oleh publik [10].

4.4. Kesimpulan

Di bab ini kita telah mengenal beberapa jenis *web service* yang tersedia di internet, yaitu Google Places dan Google Maps Directions API. Umumnya layanan-layanan tersebut memiliki parameter *request* serta *response* yang cukup banyak dan berubah-ubah, sehingga tidak memungkinkan untuk dijelaskan semuanya di buku ini.

4.5. Latihan

1. Pada penjelasan Google Places API sebelumnya, ada contoh kasus untuk mencari tempat makan yang tersedia di dekat "Monumen Nasional (Monas) Jakarta". Cobalah untuk menyusun URI yang tepat untuk mendapatkan daftar tempat makan tersebut.
2. Di contoh *response* dari Google Directions API sebelumnya, ada anggota `maneuver` pada elemen *array* `steps`. Anggota ini tidak terdokumentasi di buku ini dan di dokumentasi resminya (saat buku ini ditulis). Cobalah menganalisa kegunaan dari anggota ini, dan apa saja nilai-nilai yang mungkin dikembalikan?
3. Menggunakan Google Directions API, cobalah mencari tahu waktu yang paling baik bagi Anda untuk jalan dari rumah ke tempat kerja / sekolah / kuliah

[7] https://www.microsoft.com/cognitive-services
[8] https://developer.yahoo.com/everything.html
[9] http://docs.tiket.com/
[10] http://www.programmableweb.com/apis/directory

Anda. Salah satu caranya dengan lakukan pencarian sebanyak 24x, dengan waktu keberangkatan di jam 0 sampai 23.

5

Web Service Dalam Berbagai *Platform*

Di era sekarang, kita sering melihat sebuah aplikasi dapat diakses dalam berbagai macam *platform*. Contohnya, aplikasi *Facebook* [1] dapat diakses melalui *web browser*, Android, iPhone, dan sebagainya. Dengan memanfaatkan *web service*, berbagai macam platform dapat mengakses satu sumber data yang sama. Keuntungan lain, karena memanfaatkan protokol HTTP di port 80 komunikasi tersebut dapat dilakukan di berbagai macam jenis koneksi internet.

Di bab ini, kita akan membahas cara berkomunikasi *web service* dengan HTTP dan JSON, dalam berbagai macam *platform*. Sebagai studi kasus, kita akan memanfaatkan skenario daftar belanja yang sudah kita buat di bab 2.

Penjelasan pada subbab-subbab di bawah ini, Anda diasumsikan sudah memahami *platform* yang digunakan dan cara penggunaan *platform*-nya tidak akan dijelaskan lagi. Di setiap subbab tersedia tautan ke situs web *platform* yang digunakan, sehingga Anda dapat mempelajarinya di sana.

5.1. CodeIgniter

CodeIgniter [2] adalah salah satu framework PHP yang memanfaatkan arsitektur Model-View-Controller. Pada *web service* yang akan dibuat, kita hanya akan

[1] https://www.facebook.com
[2] https://www.codeigniter.com

memanfaatkan *controller* serta konfigurasi *routes* karena operasi-operasinya cukup sederhana dan tidak melibatkan halaman HTML.

Pertama-tama, kita lihat dulu konfigurasi *routes*, sehingga aplikasi dapat menangkap URL-URL dari *web service* Daftar Belanja dan memetakannya ke fungsi yang tepat.

/application/config/routes.php.

```
<?php
defined('BASEPATH') OR exit('No direct script access allowed');

$route['default_controller'] = 'barang'; ❶
$route['404_override'] = '';
$route['translate_uri_dashes'] = FALSE;

$route['barang']['get'] = 'barang/index'; ❷
$route['barang/(:any)']['get'] = 'barang/detail/$1';
$route['barang']['post'] = 'barang/add';
$route['barang/(:any)']['post'] = 'barang/deleteOrToggle/$1';
```

❶ Kita hanya akan memanfaatkan satu buah *controller*, yang kita namakan "Barang", dan menjadi *default controller* menggantikan *controller* contoh dari *template* CodeIgniter (walaupun sebenarnya tidak dibutuhkan, jika *client* selalu mengacu ke URL yang benar http://localhost/barang).

❷ Di sini kita mendefinisikan juga pemetaan URL yang disesuaikan dengan spesifikasi pada bab 3. Permintaan dengan metode `GET` diarahkan ke fungsi "index()" untuk mendapatkan daftar barang, atau "detail($nama)" untuk detail barang, jika diberikan nama barangnya. Permintaan dengan metode `POST` diarahkan ke fungsi "add()" untuk menambahkan barang, atau "deleteOrToggle($nama)" untuk menghapus atau mengubah status cek dari barang. Fungsi hapus dan ubah status cek dijadikan satu fungsi, karena kita tidak dapat membedakannya di *routes*. Terakhir, jika ada pemanggilan ke URL barang yang bukan merupakan salah satu dari empat URL sebelumnya, akan diarahkan ke pesan kesalahan 404 (not found).

Setelah itu, mari kita lihat *controller* yang bertugas untuk menangkap perintah dari *routes* dan memprosesnya.

/application/controllers/Barang.php.

```
<?php
```

```
defined('BASEPATH') OR exit('No direct script access allowed');

class Barang extends CI_Controller {

    public function __construct() { ❶
        parent::__construct();
        $this->load->database();
        $this->load->library('migration');
        $this->migration->latest();

        $this->output->set_content_type('application/json');
    }

    public function index() { ❷
        $this->db->select('nama, cek');
        $query = $this->db->get('daftarbelanja');

        $json_result = array(
            'status' => 'ok',
            'items' => array()
        );

        foreach ($query->result() as $row) {
            $json_result['items'][] = array(
                'cek' => $row->cek === 1,
                'nama' => $row->nama
            );
        }

        echo json_encode($json_result);
    }

    public function detail($nama) { ❸
        $nama = urldecode($nama);
        $this->db->select('nama, cek');
        $this->db->where('nama', $nama);
        $query = $this->db->get('daftarbelanja');
        $row = $query->row();
        if ($row === NULL) {
            $json_result = array(
                'status' => 'error',
                'message' => "Barang $nama tidak ditemukan!"
            );
            $this->output->set_status_header(404);
```

```
        } else {
            $json_result = array(
                'status' => 'ok',
                'item' => array(
                    'nama' => $row->nama,
                    'cek' => $row->cek === 1
                )
            );
        }
        echo json_encode($json_result);
    }

    public function add() { ❹
        $nama = $this->input->post('nama');
        $result = $this->db->insert('daftarbelanja', array(
            'nama' => $nama,
            'cek' => 0
        ));

        if ($result === TRUE) {
            $json_result = array('status' => 'ok');
            $this->output->set_status_header('201');
            $this->output->set_header('Location: /barang/$nama');
        } else {
            $json_result = array(
                'status' => 'error',
                'message' => $this->db->error()['message']
            );
            $this->output->set_status_header('500');
        }
        echo json_encode($json_result);
    }

    public function deleteOrToggle($nama) {
        $mode = $this->input->post('mode');
        switch ($mode) {
            case 'delete': ❺
                $this->db->where('nama', $nama);
                if ($this->db->delete('daftarbelanja') !== FALSE) {
                    if ($this->db->affected_rows() === 0) {
                        $json_result = array(
                            'status' => 'error',
                            'message' => "$nama tidak ditemukan"
                        );
                        $this->output->set_status_header('404');
```

```
                } else {
                    $json_result = array('status' => 'ok');
                }
            } else {
                $json_result = array(
                    'status' => 'error',
                    'message' => $this->db->error()['message']
                );
                $this->output->set_status_header('500');
            }
            break;
        case 'toggle': ❻
            $this->db->set('cek', '1-cek', FALSE);
            $this->db->where('nama', $nama);
            if ($this->db->update('daftarbelanja') !== FALSE) {
                if ($this->db->affected_rows() === 0) {
                    $json_result = array(
                        'status' => 'error',
                        'message' => "$nama tidak ditemukan",
                    );
                    $this->output->set_status_header('404');
                } else {
                    $this->db->select('nama, cek');
                    $this->db->where('nama', $nama);
                    $query = $this->db->get('daftarbelanja');
                    $row = $query->row();
                    $json_result = array(
                        'status' => 'ok',
                        'item' => array(
                            'nama' => $row->nama,
                            'cek' => $row->cek === 1
                        )
                    );
                }
            } else {
                $json_result = array(
                    'status' => 'error',
                    'message' => $this->db->error()['message']
                );
                $this->output->set_status_header('500');
            }
            break;
    }
    echo json_encode($json_result);
}
```

```
}
```

❶ Fungsi `__construct()` yang dijalankan setiap kali dipanggil akan menginisialisasi basis data serta memastikan tabel daftar belanja sudah tersedia, memanfaatkan fitur migration. Baris terakhir mengatur tipe konten yang dikirimkan menjadi `application/json`, karena kita akan selalu mengembalikan data dalam format JSON.

❷ Fungsi `index()` menangani permintaan daftar barang. Isinya tidak banyak berbeda dengan versi PHP di bab 3.

❸ Fungsi `detail($nama)` menangani permintaan detail dari sebuah barang dengan nama tertentu.

❹ Fungsi `add()` menangani permintaan untuk menambahkan barang.

❺ Bagian ini menangani permintaan untuk menghapus barang.

❻ Bagian ini menangani permintaan untuk mengubah status cek barang.

Fungsi `notfound()` menangani URL-URL yang tidak ditangkap oleh fungsi-fungsi lainnya.

Tidak seperti versi PHP standar di bab 3, akses ke basis data menggunakan fungsi-fungsi basis data yang disediakan oleh CodeIgniter sendiri [3]. Konfigurasi basis data tersebut diatur pada file konfgurasi `database.php` berikut.

/application/config/database.php.

```
<?php
defined('BASEPATH') OR exit('No direct script access allowed');
$active_group = 'default';
$query_builder = TRUE;

$db['default'] = array(
 'dsn' => '',
 'hostname' => 'localhost',
 'username' => '',
 'password' => '',
 'database' => APPPATH . 'daftarbelanja.sqlite', ❶
 'dbdriver' => 'sqlite3', ❷
 'dbprefix' => '',
 'pconnect' => FALSE,
 'db_debug' => FALSE, ❸
 'cache_on' => FALSE,
```

[3] http://www.codeigniter.com/user_guide/database/

```
 'cachedir' => '',
 'char_set' => 'utf8',
 'dbcollat' => 'utf8_general_ci',
 'swap_pre' => '',
 'encrypt' => FALSE,
 'compress' => FALSE,
 'stricton' => FALSE,
 'failover' => array(),
 'save_queries' => TRUE
);
```

❶ Di sini kita mengatur tempat menyimpan file basis data SQLite, yaitu ke file "daftarbelanja.sqlite".

❷ Kita menggunakan driver basis data sqlite versi 3.

❸ Opsi debug diset menjadi `FALSE`, untuk mencegah CodeIgniter mengeluarkan pesan kesalahan saat terjadi masalah basis data (contohnya, jika barang sudah ada saat menambahkan barang baru). Jika ada kesalahan, script kita yang akan menangani nya dan menampilkan kesalahan dalam format JSON.

Seperti disebutkan sebelumnya, kita memanfaatkan fitur *migration* pada CodeIgniter, supaya aplikasi dapat secara otomatis membetuk file basis data yang dibutuhkan jika belum ada.

Pertama-tama, kita aktifkan fitur *migration* tersebut.

/application/config/migration.php.

```
<?php
defined('BASEPATH') OR exit('No direct script access allowed');

$config['migration_enabled'] = TRUE; ❶
$config['migration_type'] = 'timestamp';
$config['migration_table'] = 'migrations';
$config['migration_version'] = 0;
$config['migration_path'] = APPPATH.'migrations/';
```

❶ Kita perlu mengubah status aktif dari migration menjadi `TRUE` sehingga migrasi diperbolehkan untuk dijalankan oleh sistem.

Setelah itu, kita juga perlu menyiapkan satu buah script migration untuk membuat file basis data tersebut.

/application/migrations/20160422100000_Initial.php.

```
<?php
defined('BASEPATH') OR exit('No direct script access allowed');

class Migration_Initial extends CI_Migration {
    public function up() {
        $this->db->query("CREATE TABLE daftarbelanja (nama TEXT PRIMARY
 KEY, cek INTEGER);");
        $this->db->query("INSERT INTO daftarbelanja VALUES ('Tomat',
 1);");
        $this->db->query("INSERT INTO daftarbelanja VALUES ('Jeruk',
 1);");
        $this->db->query("INSERT INTO daftarbelanja VALUES ('Roti',
 0);");
        $this->db->query("INSERT INTO daftarbelanja VALUES ('Sabun
 Cuci', 0);");
        $this->db->query("INSERT INTO daftarbelanja VALUES ('Sabun
 Mandi', 0);");
    }

    public function down() { }
}
```

Akhirnya, kita juga perlu membuat file `.htaccess` (jika menggunakan server Apache) atau `web.config` (jika menggunakan server IIS milik Microsoft), sehingga dapat menghilangkan bagian `index.php` pada URL.

/.htaccess.

```
RewriteEngine On
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule ^(.*)$ index.php/$1 [L]
```

/web.config.

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
<system.webServer>
    <rewrite>
        <rules>
            <rule name="Index">
                <match url="^(.*)$"/>
                <conditions>
```

```
  <add input="{REQUEST_FILENAME}" matchType="IsFile" negate="true"/>

  <add input="{REQUEST_FILENAME}" matchType="IsDirectory" negate="true"/
>
                </conditions>
                <action type="Rewrite" url="index.php/{R:1}"/>
            </rule>
        </rules>
    </rewrite>
</system.webServer>
</configuration>
```

5.2. Play Framework

Play Framework [4] adalah salah satu framework berbasis bahasa Java dan Scala yang memanfaatkan arsitektur Model-View-Controller. Pada *web service* yang akan dibuat, kita hanya akan memanfaatkan *controller* serta konfigurasi *routes* karena operasi-operasinya cukup sederhana dan tidak melibatkan halaman HTML.

Play Framework merupakan salah satu framework yang berevolusi secara cepat. Contoh di subbab ini dibuat berdasarkan Play Framework versi 2.5.2 dan Activator versi 1.3.9. Jika Anda mengunduh Play Framework versi terbaru, sintaksnya mungkin sedikit berbeda.

Pertama-tama, kita lihat dulu konfigurasi *routes*, sehingga aplikasi dapat menangkap URL-URL dari *web service* Daftar Belanja dan memetakannya ke fungsi yang tepat.

/conf/routes.

```
GET     /barang                     controllers.Barang.index ❶
GET     /barang/:nama
 controllers.Barang.detail(nama:String)
POST    /barang                     controllers.Barang.add
POST    /barang/:nama
 controllers.Barang.deleteOrToggle(nama: String)
```

[4] https://www.playframework.com

❶ Di sini kita mendefinisikan pemetaan URL yang disesuaikan dengan spesifikasi pada bab 3. Permintaan dengan metode GET diarahkan ke method "index()" milik kelas "Barang" untuk mendapatkan daftar barang, atau "detail(String nama)" untuk detail barang, jika diberikan nama barangnya. Permintaan dengan metode POST diarahkan ke method "add()" untuk menambahkan barang, atau "deleteOrToggle(String nama)" untuk menghapus atau mengubah status cek dari barang. Fungsi hapus dan ubah status cek dijadikan satu fungsi, karena kita tidak dapat membedakannya di *routes*.

Setelah itu, mari kita lihat kelas *controller* "Barang" yang bertugas untuk menangkap perintah dari *routes* dan memprosesnya.

/app/controllers/Barang.java.

```
package controllers;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import javax.inject.Inject;

import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.node.ArrayNode;
import com.fasterxml.jackson.databind.node.ObjectNode;

import play.data.DynamicForm;
import play.data.FormFactory;
import play.db.*;
import play.libs.Json;
import play.mvc.Controller;
import play.mvc.Result;

public class Barang extends Controller {
    @Inject ❶
    Database db;
    @Inject
    FormFactory ff;

    public Result index() throws SQLException { ❷
        Connection connection = null;
```

```
        try {
            connection = db.getConnection();
            Statement statement = connection.createStatement();
            ResultSet dbResult = statement.executeQuery("SELECT nama,
cek FROM daftarbelanja");
            ArrayNode items = Json.newArray();
            while (dbResult.next()) {
                ObjectNode item = Json.newObject();
                item.put("nama", dbResult.getString(1));
                item.put("cek", dbResult.getBoolean(2));
                items.add(item);
            }
            ObjectNode result = Json.newObject();
            result.put("status", "ok");
            result.set("items", items);
            connection.close();
            return ok(result);
        } catch (Exception e) {
            return
internalServerError(generateErrorJson(e.getMessage()));
        } finally {
            connection.close();
        }
    }

    public Result detail(String nama) throws SQLException { ❸
        Connection connection = null;
        try {
            connection = db.getConnection();
            PreparedStatement statement =
connection.prepareStatement("SELECT nama, cek FROM daftarbelanja WHERE
nama=?");
            statement.setString(1, nama);
            ResultSet dbResult = statement.executeQuery();
            if (dbResult.next()) {
                ObjectNode item = Json.newObject();
                item.put("nama", dbResult.getString(1));
                item.put("cek", dbResult.getBoolean(2));
                ObjectNode result = Json.newObject();
                result.put("status", "ok");
                result.set("item", item);
                return ok(result);
            } else {
                return notFound(generateErrorJson("Barang " + nama + "
tidak ditemukan"));
```

```
            }
        } catch (Exception e) {
            return
internalServerError(generateErrorJson(e.getMessage()));
        } finally {
            connection.close();
        }
    }

    public Result add() throws SQLException { ❹
        Connection connection = null;
        try {
            DynamicForm form = ff.form().bindFromRequest();
            String nama = form.get("nama");
            connection = db.getConnection();
            PreparedStatement statement =
connection.prepareStatement("INSERT INTO daftarbelanja (nama, cek)
VALUES (?,?)");
            statement.setString(1, nama);
            statement.setBoolean(2, false);
            int affectedRows = statement.executeUpdate();
            if (affectedRows >= 1) {
                ObjectNode result = Json.newObject();
                result.put("status", "ok");
                response().setHeader("Location", "/barang/ " + nama);
                return created(result);
            } else {
                return internalServerError(generateErrorJson("Barang " +
nama + " sudah ada di daftar belanja!"));
            }
        } catch (Exception e) {
            return
internalServerError(generateErrorJson(e.getMessage()));
        } finally {
            connection.close();
        }

    }

    public Result deleteOrToggle(String nama) throws SQLException {
        Connection connection = null;
        try {
            connection = db.getConnection();
            DynamicForm form = ff.form().bindFromRequest();
            String mode = form.get("mode");
```

```
            PreparedStatement statement;
            switch (mode) {
            case "delete": ❺
                statement = connection.prepareStatement("DELETE FROM
daftarbelanja WHERE nama=?");
                statement.setString(1, nama);
                if (statement.executeUpdate() >= 1) {
                    ObjectNode result = Json.newObject();
                    result.put("status", "ok");
                    return ok(result);
                } else {
                    return notFound(generateErrorJson(nama + " tidak
ditemukan"));
                }
            case "toggle": ❻
                statement = connection.prepareStatement("UPDATE
daftarbelanja SET cek=1-cek WHERE nama=?");
                statement.setString(1, nama);
                if (statement.executeUpdate() >= 1) {
                    statement = connection.prepareStatement("SELECT
nama, cek FROM daftarbelanja WHERE nama=?");
                    statement.setString(1, nama);
                    ResultSet dbResult = statement.executeQuery();
                    dbResult.next();
                    ObjectNode item = Json.newObject();
                    item.put("nama", dbResult.getString(1));
                    item.put("cek", dbResult.getBoolean(2));

                    ObjectNode result = Json.newObject();
                    result.put("status", "ok");
                    result.set("item",  item);

                    return ok(result);
                } else {
                    return notFound(generateErrorJson(nama + " tidak
ditemukan"));
                }
            default:
                return notFound();
            }
        } catch (Exception e) {
            e.printStackTrace();
            return
internalServerError(generateErrorJson(e.getMessage()));
        } finally {
```

```
            connection.close();
        }

    }

    private JsonNode generateErrorJson(String message) { ❼
        ObjectNode node = Json.newObject();
        node.put("status", "error");
        node.put("message", message);
        return node;
    }
}
```

❶ Ada dua atribut hasil injeksi yang digunakan di kelas ini, yaitu "db" untuk koneksi basis data dan "ff" yang akan digunakan untuk menerima parameter dari *request* POST.

❷ Method `index()` menangani permintaan daftar barang. Secara sederhana, method ini melakukan *query* ke basis data untuk mendapatkan daftar barang, dan menuliskannya dalam format JSON.

❸ Method `detail(String nama)` menangani permintaan detail dari sebuah barang dengan nama tertentu. Method ini juga melakukan *query* ke basis data dan menuliskannya kembali dalam format JSON.

❹ Method `add()` menangani permintaan untuk menambahkan barang, di mana nama barang didapatkan dari parameter *request* POST. Method ini melakukan *query* update ke basis data dan mengembalikan perintah *redirect* ke barang yang baru dibuat.

❺ Bagian ini menangani permintaan untuk menghapus barang.

❻ Bagian ini menangani permintaan untuk mengubah status cek barang.

❼ Method "generateErrorJSON(String message)" merupakan method bantuan untuk membuat objek JSON yang menyatakan pesan kesalahan.

Untuk mengakses basis data, kita juga perlu mengkonfigurasi basis data yang akan digunakan. Di sini kita akan menggunakan basis data standar Play Framework, yaitu H2. Selain itu, kita juga akan memanfaatkan fitur evolution untuk membuat basis data awal jika belum ada. Untuk itu, kita perlu mengubah file konfigurasi `application.conf` seperti berikut.

/conf/application.conf.

```
# Bagian lain dari file konfigurasi tidak dituliskan di sini karena
 terlalu
```

```
# panjang

play.evolutions { ❶
  autoApply = true
}

db { ❷
  default.driver = org.h2.Driver
  default.url = "jdbc:h2:mem:play"
  default.username = sa
  default.password = ""
}
```

❶ Di sini kita mengatur Play Framework untuk melakukan evolusi (membuat basis data) secara otomatis jika belum ada.

❷ Kita mengatur aplikasi untuk menggunakan basis data H2, di mana isi basis data disimpan di memori.

Kita juga perlu mengaktifkan *library* evolusi di file `build.sbt` seperti berikut.

/build.sbt.

```
name := """DaftarBelanja"""

version := "1.0-SNAPSHOT"

lazy val root = (project in file(".")).enablePlugins(PlayJava)

scalaVersion := "2.11.7"

libraryDependencies ++= Seq(
  javaJdbc,
  cache,
  javaWs,
  evolutions ❶
)
```

❶ Kita tambahkan *library* `evolutions` di sini.

Terakhir, kita buat *script* SQL untuk membuat basis data awal.

/conf/evolutions/default/1.sql.

```
# --- !Ups
```

```
CREATE TABLE daftarbelanja (nama VARCHAR PRIMARY KEY, cek INTEGER);
INSERT INTO daftarbelanja VALUES ('Tomat', 1);
INSERT INTO daftarbelanja VALUES ('Jeruk', 1);
INSERT INTO daftarbelanja VALUES ('Roti', 0);
INSERT INTO daftarbelanja VALUES ('Sabun Cuci', 0);
INSERT INTO daftarbelanja VALUES ('Sabun Mandi', 0);

# --- !Downs

DROP TABLE daftarbelanja;
```

Jika Anda menjalankan *web service* ini dengan `activator run`, pastikan bahwa *client* Anda terkoneksi ke port 9000 (default Play Framework) bukan 80 (default HTTP).

5.3. jQuery

Salah satu pemanfaatan web service yang pertama muncul adalah AJAX (Asynchronous JavaScript and XML), di mana sebuah halaman web dapat memperbaharui kontennya tanpa harus melakukan *refresh* halaman.

jQuery [5] adalah salah satu *library* yang menyederhanakan operasi AJAX dan manipulasi objek di HTML. Kita dapat membuat *client* dari *web service* yang telah kita buat dengan kode HTML berikut.

/index.html.

```
<!doctype html>
<html>
  <head>
    <title>Daftar Belanja</title>
    <style type="text/css"> ❶
      ul {
        list-style-type: none;
        padding-left: 0px;
      }
      button {
        border: none;
        padding: 2px;
        margin-left: 5px;
        text-transform: uppercase;
```

[5] https://jquery.com/

```
        cursor: pointer;
      }
    </style>
  </head>
  <body>
    <ul id="baranglist"> ❷
    </ul>
    <label for="newitem">Tambah barang: </label> ❸
    <input type="text" id="newitem"/>
    <button id="addButton">Tambah</button>
    <script src="js/jquery-2.2.3.min.js"></script>
    <script>
      $(document).ready(function () {
        function refresh() { ❹
          $.getJSON('/barang/').done(function (data) {
            var list = $('#baranglist');
            list.empty();
            $.each(data.items, function (key, value) {
              var checkbox = $('<input type="checkbox" id="' +
value.nama + '"/>'); ❺
              if (value.cek) {
                checkbox.attr('checked', '');
              }
              var label = $('<label for="' + value.nama + '">' +
value.nama + '</label>');
              var deleteButton = $('<button>delete</button>');
              var li = $('<li></li>');
              li.append(checkbox).append(label).append(deleteButton);
              list.append(li);

              checkbox.change(function () { ❻
                $.post('/barang/' + value.nama, {mode: 'toggle'});
              });


              deleteButton.click(function () { ❼
                $.post('/barang/' + value.nama, {mode:
'delete'}).done(function () {
                  refresh();
                });
              });
            });
          });
        }
```

```
        $('#addButton').click(function () { ❽
          $.post('/barang/', {mode: 'add', nama:
 $('#newitem').val()}).done(function () {
            refresh();
          }).fail(function() {
            alert('Error');
          });
        });

        refresh(); ❾
      });
    </script>
  </body>
</html>
```

❶ Di sini kita mengatur beberapa *styles* menggunakan CSS. Walaupun tidak ada efeknya ke *web service*, *styling* ini membuat tampilan aplikasi lebih mudah dimengerti.

❷ Kita siapkan daftar tidak terurut dalam bentuk sebuah tag `ul` dengan id "baranglist" untuk nantinya menampilkan daftar barang belanjaan. Saat ini daftar tersebut kosong.

❸ Tiga tag HTML `label`, `input`, dan `button` berfungsi sebagai elemen UI yang bertugas untuk menerima perintah "Tambah Barang" dari pengguna.

❹ Kita buat fungsi "refresh()" yang jika dijalankan akan mengisi daftar "baranglist" dengan daftar barang yang didapatkan dari *server* memanfaatkan *web service*. Fungsi ini mengirimkan HTTP request ke URL "/barang".

❺ Saat *response* sudah diterima dalam bentuk JSON, kita mengkonstruksi HTML yang diperlukan untuk mengisi daftar "baranglist". Masing-masing berisi `checkbox`, `label` serta `button` untuk menghapus.

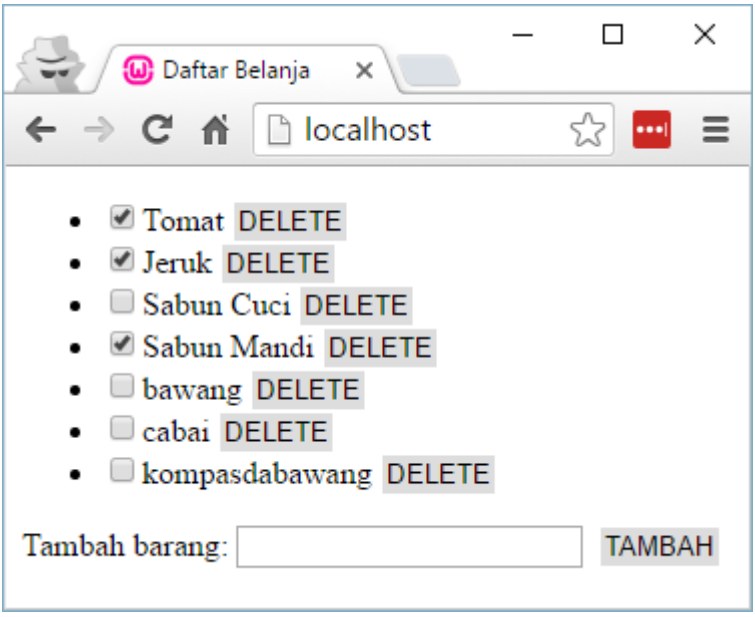
❻ Kita perlu menambahkan *event handler* pada `checkbox` tersebut, supaya jika nilainya berubah aplikasi mengirimkan permintaan untuk mengubah status cek barang tersebut pada *server*.

❼ Kita juga perlu menambahkan *event handler* pada tombol hapus, supaya jika ditekan aplikasi mengirimkan permintaan untuk menghapus barang tersebut, dan melakukan *refresh* (karena ada satu barang yang telah hilang).

❽ Kita juga menambahkan *event handler* untuk tombol tambah, supaya jika ditekan aplikasi mengirimkan permintaan untuk menambahkan barang baru.

❾ Terakhir, kita memanggil fungsi "refresh()" yang telah kita buat saat aplikasi dibuka, sehingga daftar belanja terisi dengan data yang ada di server.

Aplikasi tersebut jika dijalankan akan menghasilkan tampilan kurang lebih seperti di bawah ini.



Gambar 5.1. Aplikasi Daftar Belanja dengan jQuery

5.4. Android

Android merupakan salah satu platform peranti bergerak yang paling banyak ditemui. Untuk membuat aplikasi berbasis Android, kita menggunakan Android SDK dan Android Studio [6]. Pemanfaatan *web service* untuk aplikasi peranti bergerak sangat efektif, karena peranti dapat terkoneksi ke internet melalui berbagai macam jaringan. Peranti dapat terhubung ke internet melalui koneksi jaringan telepon (3G/4G) maupun berbagai macam wifi. Untuk itu, diperlukan metode komunikasi yang paling memungkinkan untuk berjalan di segala kondisi. Seperti dijelaskan di bab 2, *web service* memanfaatkan teknologi yang sama dengan akses situs web yang sudah sangat umum, kita dapat cukup yakin bahwa komunikasi tersebut dapat berjalan di berbagai kondisi.

[6] http://developer.android.com/

Sama dengan contoh Aplikasi Android yang kita buat memanfaatkan *library* jsoup. Oleh karena itu, kita perlu menambahkan kebutuhan tersebut di konfigurasi gradle.

/app/build.gradle.

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 23
    buildToolsVersion "23.0.3"

    defaultConfig {
        applicationId "chapter5.daftarbelanja"
        minSdkVersion 15
        targetSdkVersion 23
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-
android.txt'), 'proguard-rules.pro'
        }
    }
}

dependencies {
    compile fileTree(include: ['*.jar'], dir: 'libs')
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:appcompat-v7:23.3.0'
    compile 'com.android.support:design:23.3.0'
    compile 'org.jsoup:jsoup:1.9.1' ❶
}
```

❶ Kita tambahkan kebutuhan *library* jsoup di baris ini.

Karena aplikasi kita akan berkomunikasi dengan *server*, maka kita juga perlu menambahkan permintaan izin untuk menggunakan internet di konfigurasi manifes.

/app/src/main/AndroidManifest.xml.

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="chapter5.daftarbelanja">

    <uses-permission android:name="android.permission.INTERNET"/> ❶

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name"
            android:theme="@style/AppTheme.NoActionBar">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />


  <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

❶ Kita tambahkan permintaan untuk izin penggunaan internet.

Mari kita masuk ke kode program aplikasi. Pertama-tama, kita buat kelas "Barang" yang akan merepresentasikan barang belanjaan. Kelas ini memiliki dua atribut yaitu nama (nama barang) dan cek (status apakah sudah dibeli atau belum).

/app/src/main/java/chapter5/daftarbelanja/Barang.java.

```
package chapter5.daftarbelanja;

public class Barang {
  public String nama;
  public boolean cek;

  public Barang(String nama, boolean cek) {
    this.nama = nama;
    this.cek = cek;
  }
}
```

Setelah itu, kita siapkan adapter untuk kelas barang dalam kelas "Adapter". Adapter ini berfungsi untuk menghubungkan objek barang dengan elemen UI `ListView` milik Android. Selain itu, kelas ini juga berfungsi untuk berkomunikasi dengan *server web service*.

/app/src/main/java/chapter5/daftarbelanja/BarangAdapter.java.

```
package chapter5.daftarbelanja;

import android.content.Context;
import android.os.AsyncTask;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.CheckBox;
import android.widget.CompoundButton;
import android.widget.ImageButton;
import android.widget.Toast;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;
import org.jsoup.Connection;
import org.jsoup.helper.HttpConnection;

import java.util.ArrayList;
import java.util.List;

public class BarangListAdapter extends BaseAdapter {

  public static String BASE_URL = "http://10.0.2.2"; ❶

  List<Barang> daftarBelanja; ❷
  Context context; ❸

  public BarangListAdapter(Context context) {
    this.daftarBelanja = new ArrayList<>();
    this.context = context;
  }

  @Override
  public int getCount() {
    return this.daftarBelanja.size();
  }
```

```
 @Override
 public Object getItem(int i) {
   return this.daftarBelanja.get(i);
 }

 @Override
 public long getItemId(int i) {
   return i;
 }

 @Override
 public View getView(int position, View convertView, ViewGroup parent)
{ ❹
   View view = convertView == null ? View.inflate(this.context,
R.layout.view_barang, null) : convertView;
   final CheckBox barangCheckbox = (CheckBox)
view.findViewById(R.id.barang_checkbox);
   ImageButton deleteBarangButton = (ImageButton)
view.findViewById(R.id.barang_delete);
   final Barang barang = this.daftarBelanja.get(position);
   barangCheckbox.setOnCheckedChangeListener(null);
   barangCheckbox.setText(barang.nama);
   barangCheckbox.setChecked(barang.cek);
   barangCheckbox.setOnCheckedChangeListener(new
CompoundButton.OnCheckedChangeListener() {
     @Override
     public void onCheckedChanged(CompoundButton buttonView, boolean
isChecked) {
       toggleBarang(barangCheckbox.getText());
     }
   });
   deleteBarangButton.setOnClickListener(new View.OnClickListener() {
     @Override
     public void onClick(View v) {
       deleteBarang(barangCheckbox.getText());
     }
   });
   return view;
 }

 public void refreshList() { ❺
   AsyncTask<BarangListAdapter, Object, JSONObject> listRequest = new
AsyncTask<BarangListAdapter, Object, JSONObject>() {
     BarangListAdapter parent;
```

```
      Exception savedException;

      @Override
      protected JSONObject doInBackground(BarangListAdapter... parents)
 {
        this.parent = parents[0];
        try {
          Connection connection = HttpConnection.connect(BASE_URL + "/
barang");
          connection.ignoreHttpErrors(true);
          connection.ignoreContentType(true);
          String content = connection.execute().body();
          return new JSONObject(content);
        } catch (Exception e) {
          savedException = e;
          return null;
        }
      }

      @Override
      protected void onPostExecute(JSONObject result) {
        if (result != null) {
          try {
            JSONArray items = result.getJSONArray("items");
            parent.daftarBelanja.clear();
            for (int i = 0; i < items.length(); i++) {
              JSONObject row = items.getJSONObject(i);
              parent.daftarBelanja.add(new Barang(row.getString("nama"),
 row.getBoolean("cek")));
            }
            parent.notifyDataSetChanged();
          } catch (Exception e) {
            savedException = e;
          }
        }
        if (savedException != null) {
          reportError(savedException);
        }
      }
    };
    listRequest.execute(this);
  }

  public void addBarang(final CharSequence nama) { ❻
```

```
    AsyncTask<BarangListAdapter, Object, JSONObject> addRequest = new
 AsyncTask<BarangListAdapter, Object, JSONObject>() {
      BarangListAdapter parent;
      Exception savedException;

      @Override
      protected JSONObject doInBackground(BarangListAdapter... parents)
 {
        this.parent = parents[0];
        try {
          Connection connection = HttpConnection.connect(BASE_URL + "/
barang/");
          connection.method(Connection.Method.POST);
          connection.data("mode", "add");
          connection.data("nama", nama.toString());
          connection.followRedirects(false);
          connection.ignoreHttpErrors(true);
          connection.ignoreContentType(true);
          String content = connection.execute().body();
          return new JSONObject(content);
        } catch (Exception e) {
          savedException = e;
          return null;
        }
      }

      @Override
      protected void onPostExecute(JSONObject result) {
        if (result != null) {
          try {
            if (result.getString("status").equals("ok")) {
              refreshList();
            } else {
              savedException = new
 JSONException(result.getString("message"));
            }
          } catch (JSONException e) {
            savedException = e;
          }
        }
        if (savedException != null) {
          reportError(savedException);
        }
      }
    };
    addRequest.execute(this);
```

```
  }

  public void toggleBarang(final CharSequence nama) { ❼
    AsyncTask<BarangListAdapter, Object, JSONObject> toggleRequest = new
 AsyncTask<BarangListAdapter, Object, JSONObject>() {
      BarangListAdapter parent;
      Exception savedException;

      @Override
      protected JSONObject doInBackground(BarangListAdapter... parents)
 {
        this.parent = parents[0];
        try {
          Connection connection = HttpConnection.connect(BASE_URL + "/
barang/" + nama);
          connection.method(Connection.Method.POST);
          connection.data("mode", "toggle");
          connection.ignoreHttpErrors(true);
          connection.ignoreContentType(true);
          String content = connection.execute().body();
          return new JSONObject(content);
        } catch (Exception e) {
          savedException = e;
          return null;
        }
      }

      @Override
      protected void onPostExecute(JSONObject result) {
        if (result != null) {
          try {
            if (!result.getString("status").equals("ok")) {
              savedException = new
 JSONException(result.getString("message"));
            }
          } catch (JSONException e) {
            savedException = e;
          }
        }
        if (savedException != null) {
          reportError(savedException);
        }
      }
    };
    toggleRequest.execute(this);
```

```
  }

  public void deleteBarang(final CharSequence nama) { ❽
    AsyncTask<BarangListAdapter, Object, JSONObject> deleteRequest = new
 AsyncTask<BarangListAdapter, Object, JSONObject>() {
      BarangListAdapter parent;
      Exception savedException;

      @Override
      protected JSONObject doInBackground(BarangListAdapter... parents)
 {
        this.parent = parents[0];
        try {
          Connection connection = HttpConnection.connect(BASE_URL + "/
barang/" + nama);
          connection.method(Connection.Method.POST);
          connection.data("mode", "delete");
          connection.ignoreHttpErrors(true);
          connection.ignoreContentType(true);
          String content = connection.execute().body();
          return new JSONObject(content);
        } catch (Exception e) {
          savedException = e;
          return null;
        }
      }

      @Override
      protected void onPostExecute(JSONObject result) {
        if (result != null) {
          try {
            if (result.getString("status").equals("ok")) {
              refreshList();
            } else {
              savedException = new
 JSONException(result.getString("message"));
            }
          } catch (JSONException e) {
            savedException = e;
          }
        }
        if (savedException != null) {
          reportError(savedException);
        }
      }
```

```
    };
    deleteRequest.execute(this);
  }

  public void reportError(Exception e) { ❾
    Toast toast = Toast.makeText(this.context, e.getMessage(),
 Toast.LENGTH_SHORT);
    toast.show();
    e.printStackTrace();
  }
}
```

❶ Di sini kita siapkan konstanta URL dari *server web service* yang digunakan. Jika menggunakan *emulator*, nilainya dapat diisi dengan http://10.0.2.2 untuk mengacu ke komputer tempat emulator ini dijalankan.

❷ Kita siapkan atribut "daftarBelanja" yang menyimpan daftar dari barang belanjaan. Atribut ini diinisialisasi di konstruktor.

❸ Kita juga menyiapkan atribut bertipe `Context` yang akan kita gunakan jika ingin berkomunikasi dengan *Activity* yang menggunakan *adapter* ini.

❹ Dari beberapa method yang harus diimplementasikan karena kelas diturunkan dari kelas `BaseAdapter`, method `getView(int position, View convertView, ViewGroup parent)` lah yang cukup kompleks. Di sini kita menyiapkan elemen-elemen UI untuk barang, diambil dari layout "view_barang" (dijelaskan isinya di bawah). Elemen-elemen ini kemudian akan diisi dengan data untuk barang yang dimaksud, serta menambahkan *event handler* untuk perubahan status cek dan tombol hapus. Method ini juga dilengkapi dengan perintah untuk memanfaatkan view sebelumnya, jika sudah pernah ada.

❺ Method "refreshList()" adalah method yang kita buat untuk mengirimkan perintah untuk mendapatkan daftar barang, dan menampilkan hasilnya ke `ListView` kita. Sesungguhnya method ini sesederhana mengirimkan *request* `GET` ke *server* dan mengolah hasilnya untuk memperbaharui atribut daftarBelanja. Namun, karena Android SDK mensyaratkan *request* dilakukan pada *thread* terpisah, maka kita perlu memanfaatkan kelas `AsyncTask`.

❻ Method "addBarang()" digunakan untuk mengirimkan perintah menambahkan barang ke *server*. Seperti sebelumnya, method ini juga dirumitkan dengan penggunaan `AsyncTask`.

❼ Method "toggleBarang()" digunakan untuk mengirimkan perintah mengubah status cek barang ke *server*. Method ini juga menggunakan `AsyncTask`.

❽ Method "deleteBarang()" digunakan untuk mengirimkan perintah menghapus barang ke *server*. Method ini juga menggunakan `AysncTask`.

❾ Method "reportError(Exception e)" adalah method bantuan yang dipanggil pada method-method lainnya, untuk menampilkan pesan kesalahan ke pengguna. Pesan kesalahan ini ditampilkan dalam bentuk *toast message*.

Layout "view_barang" yang digunakan pada kode sebelumnya direpresentasikan dengan kode XML seperti berikut.

/app/src/main/res/layout/view_barang.xml.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/
android"
                android:layout_width="match_parent"
                android:layout_height="match_parent">

  <ImageButton
    android:id="@+id/barang_delete"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentEnd="true"
    android:layout_alignParentRight="true"
    android:layout_alignParentTop="true"
    android:contentDescription="@string/delete"
    android:src="@android:drawable/ic_delete"/>


  <CheckBox
    android:id="@+id/barang_checkbox"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_marginTop="8dip"
    android:layout_toLeftOf="@id/barang_delete"
    android:layout_toStartOf="@id/barang_delete"
    android:text="@string/barang"/>

</RelativeLayout>
```

Activity utama dari aplikasi kita direpresentasikan dengan kelas "MainActivity", yang isinya adalah seperti di bawah ini.

/app/src/main/java/chapter5/daftarbelanja/MainActivity.java.

```
package chapter5.daftarbelanja;

import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.EditText;
import android.widget.ListView;

public class MainActivity extends AppCompatActivity {

  BarangListAdapter barangListAdapter; ❶

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);

    barangListAdapter = new BarangListAdapter(this); ❷
    ListView barangListView = (ListView)
 findViewById(R.id.barang_list_view);
    barangListView.setAdapter(barangListAdapter);
    barangListAdapter.refreshList();
  }

  @Override
  public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is
 present.
    getMenuInflater().inflate(R.menu.menu_main, menu);
    return true;
  }

  @Override
  public boolean onOptionsItemSelected(MenuItem item) { ❸
    int id = item.getItemId();

    //noinspection SimplifiableIfStatement
    if (id == R.id.action_refresh) {
```

```
      barangListAdapter.refreshList();
      return true;
    }

    return super.onOptionsItemSelected(item);
  }

  public void onAddClicked(View view) { ❹
    EditText addBarangTextView = (EditText)
 findViewById(R.id.add_barang_edit_text);
    barangListAdapter.addBarang(addBarangTextView.getText());
  }
}
```

❶ Kita siapkan dulu atribut bertipe BarangListAdapter untuk menyimpan barang-barang kita, dan menghubungkannya dengan `ListView`.

❷ Di sini kita inisialisasi adapter serta `ListView` kita, dan melakukan *refresh* satu kali untuk mengisi daftar barang dari *server*.

❸ Kita juga siapkan *event handler* jika tombol *refresh* ditekan pada menu tersebut.

❹ Kita juga memanggil fungsi "addBarang()" jika tombol *add* tersebut ditekan.

5.5. Kesimpulan

Salah satu keuntungan *web service* adalah kompatibel dengan banyak *platform*. Ditambah lagi dengan tersedianya *library-library* di masing-masing *platform*, memudahkan pembangunan aplikasi yang memanfaatkan *web service multiplatform*. Di bab ini kita mencoba mengimplementasikan *web service* di dua *platform* server dan dua *platform client*.

5.6. Latihan

1. CodeIgniter dan Play Framework merupakan dua buah *framework server* yang masih cukup aktif dikembangkan (saat buku ini ditulis), sehingga mendukung *web service* berarsitektur REST. Cobalah untuk mengubah kode tersebut sehingga menggunakan metode `PUT` untuk operasi menambah barang dan `DELETE` untuk operasi menghapus barang (seperti pada latihan no. 3 di bab 3).

2. Apakah Anda terbiasa membuat aplikasi pada *platform* lain? Cobalah membuat aplikasi yang kompatibel dengan *server* atau *client* Daftar Belanja pada *platform* tersebut! Beberapa ide: NodeJS (*server*), Ruby on Rails (*server*), iPhone (*client*), atau .NET (*client*).
3. Sampai di sini, Anda sudah belajar seluruh bahan yang ditawarkan buku ini dalam membangun aplikasi yang memanfaatkan *web service*. Cobalah mencari ide dan bangun aplikasi milik Anda sendiri!

Daftar Pustaka

ECMA International "The JSON Data Interchange Format." ECMA-404, 2013.

R. Fielding, J. Reschke "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing." RFC 7230 (Proposed Standard), 2014.

R. Fielding, J. Reschke "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content." RFC 7231 (Proposed Standard), 2014.

R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures." University of California, 2000.

T. Bray "The JavaScript Object Notation (JSON) Data Interchange Format." RFC 7159 (Proposed Standard), 2014.