

Contoh dan Penjelasan BAHASA SINGKONG

Aplikasi web dan topik lanjutan

Termasuk pembahasan:
JavaScript Object Notation (JSON) dan HTTP client

```
if (request_method() == "GET") {
    var g = cgi_get()
    var error = ""
    if (g["error"] == "auth") {
        var error = "Authentication failed"
    }
    cgi_header()
    print("
<!DOCTYPE html>
<html lang='en'>
<head>
    <title>Login</title>
</head>
<body>
    " + error +
    "
        <form action='login.web' method='post'>
            Username:<input type='text' name='u'>
            Password:<input type='password' name='p'>
            <input type='submit' value='login'>
        </form>
    </body>
</html>
")
    exit()
}
if (request_method() == "POST") {
    var form = cgi_post()
    var u = form["u"]
    var p = form["p"]
    if (u == "admin" & p == "admin") {
        var sess = session_new()
        set(sess, "user", u)
        session_file_set(sess_path, sess)
        cgi_header(
            header_session(sess) +
            header_location("home.web")
        )
    } else {
        cgi_header(
            header_location("login.web?error=auth")
        )
    }
    exit()
}
```



Dr. Noprianto
Dr. Buyung Sofiaro Munir
Dr. Sarwo

Contoh dan Penjelasan Bahasa Singkong: Aplikasi Web dan Topik Lanjutan

Penulis: Dr. Noprianto, Dr. Buyung Sofiarso Munir, Dr. Sarwo

ISBN: 978-602-52770-6-1

Penerbit:

PT. Stabil Standar Sinergi

Alamat	Puri Indah Financial Tower Lantai 6, Unit 0612 Jl. Puri Lingkar Dalam Blok T8, Puri Indah, Kembangan, Jakarta Barat 11610
Website	www.singkong.dev
Email	info@singkong.dev

Cover buku:

- Masakan: singkong goreng oleh Meike Thedy, S.Kom.
- Desain cover oleh Noprianto, menggunakan GIMP. Filter yang digunakan adalah: Mosaic. Font yang digunakan adalah Calibri. Masing-masing teks tulisan (kecuali tulisan Penerbit) pada cover dibuat dengan beberapa layer untuk mendapatkan efek outline.
- Cuplikan kode adalah bagian dari contoh dalam buku.

Hak cipta dilindungi undang-undang.

Daftar Isi

Kata Pengantar	2
Persiapan	3
Instalasi dan Konfigurasi HTTP Server	5
HTTP dan CGI	19
Request Method GET dan Query String	27
Form: GET	33
Form: POST	37
Memeriksa Request Method	41
Bekerja dengan Session	47
Mengenal format data-interchange JSON	57
Form: POST (HASH)	63
Contoh HTTP Request dengan JSON	69
HTTP Client	75
Daftar Pustaka	85

Kata Pengantar

Buku ini berisi sejumlah contoh source code dan penjelasan langkah demi langkah yang mudah dipahami untuk membuat aplikasi web, dengan bahasa pemrograman Singkong.

Contoh-contoh yang dibahas mencakup instalasi dan konfigurasi HTTP server, pengenalan HTTP dan CGI, menangani request GET dan POST, bekerja dengan session, JavaScript Object Notation (JSON), dan HTTP client.

Jakarta, Februari 2024

Tim penulis

Untuk referensi dan dokumentasi lengkap bahasa Singkong, Anda mungkin ingin membaca buku-buku gratis berikut:

- Menegal dan Menggunakan Bahasa Pemrograman Singkong (ISBN: 978-602-52770-1-6, Dr. Noprianto).
- Contoh dan Penjelasan Bahasa Singkong: Dasar-dasar Aplikasi GUI (ISBN: 978-602-52770-3-0, Dr. Noprianto, Dr. Karto Iskandar, Benfano Soewito, Ph.D.).
- Contoh dan Penjelasan Bahasa Singkong: Mahir Bekerja dengan GUI (ISBN: 978-602-52770-4-7, Dr. Noprianto, Dr. Wartika, Dr. Ford Lumban Gaol).
- Contoh dan Penjelasan Bahasa Singkong: Bekerja dengan Database Relasional (ISBN: 978-602-52770-5-4, Dr. Noprianto, Dr. Maria Seraphina Astriani, Dr. Fredy Purnomo).

Semua buku tersebut juga dapat dibaca dengan mengunjungi:
<https://singkong.dev>

Persiapan

Siapkanlah sebuah komputer, yang dilengkapi layar, keyboard, dan mouse/trackpad. Untuk perangkat keras komputernya, dapat menggunakan spesifikasi komputer mulai dari yang terbaru ataupun yang telah dijual sekitar 20 tahun yang lalu. Yang penting, dapat menjalankan salah satu dari daftar sistem operasi berikut.

Interpreter Singkong (dan program yang Anda buat nantinya) dapat berjalan pada berbagai sistem operasi berikut:

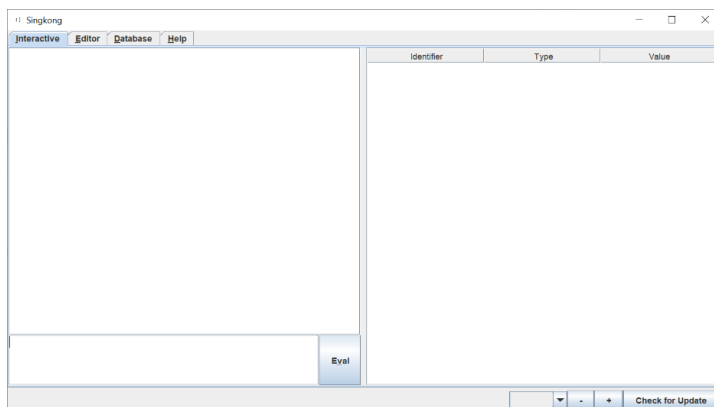
- macOS (mulai dari Mac OS X 10.4 Tiger)
- Windows (mulai dari Windows 98)
- Linux (mulai yang dirilis sejak awal 2000-an; juga termasuk Raspberry Pi OS dan Debian di Android)
- Chrome OS (sejak tersedia Linux development environment, telah diuji pada versi 101 di chromebook)
- Solaris (telah diuji pada versi 11.4)
- FreeBSD (telah diuji pada versi 13.0 dan 12.1)
- OpenBSD (telah diuji pada versi 7.0 dan 6.6)
- NetBSD (telah diuji pada versi 9.2 dan 9.0)

(Khusus untuk buku ini, pastikanlah HTTP Server yang mendukung CGI tersedia di sistem operasi yang Anda gunakan—kita akan bahas ini di bab tahapan instalasi dan konfigurasi HTTP Server.)

Setelah komputer siap, lakukanlah instalasi Java, apabila belum terinstal sebelumnya. Secara teknis, Anda hanya membutuhkan Java Runtime Environment, versi 5.0 atau lebih baru. Versi 5.0 dirilis pada tahun 2004 (sekitar 20 tahun lalu pada saat buku ini ditulis). Gunakan versi yang masih didukung secara teknis, apabila memungkinkan. (Kenapa perlu menginstalasi Java? Karena interpreter Singkong ditulis dengan bahasa Java dan bahasa Singkong itu sendiri.)

Sebagai langkah terakhir, downloadlah interpreter Singkong, yang akan selalu didistribusikan sebagai file jar tunggal (Singkong.jar). Downloadlah selalu dari <https://nopri.github.io/Singkong.jar>. Pada saat buku ini ditulis, ukurannya hanya 4,3 MB dan berisikan semua yang diperlukan untuk mengikuti semua contoh dalam buku ini. Pastikanlah Anda menggunakan versi terbaru.

Apabila Singkong.jar telah dapat dijalankan, lanjutkanlah ke instalasi dan konfigurasi HTTP Server. Beberapa contoh dalam buku ini perlu diketikkan pada tab Interactive untuk menguji kode program secara langsung. Editor yang disertakan dapat digunakan untuk mengetikkan, menyimpan/membuka, dan menjalankan kode program yang lebih panjang, walau Anda mungkin ingin menggunakan editor lain yang lebih nyaman.



Apabila langkah detil instalasi Java dan menjalankan Singkong.jar diperlukan, bacalah juga halaman 5 pada buku gratis: *Contoh dan Penjelasan Bahasa Singkong: Dasar-dasar aplikasi GUI*.

Dan, apabila Anda perlu mendistribusikan program yang Anda buat dengan bahasa Singkong dalam satu file jar yang dapat dijalankan, bacalah juga bab Distribusi Aplikasi pada buku gratis: *Mengenal dan Menggunakan Bahasa Pemrograman Singkong*.

Instalasi dan Konfigurasi HTTP Server

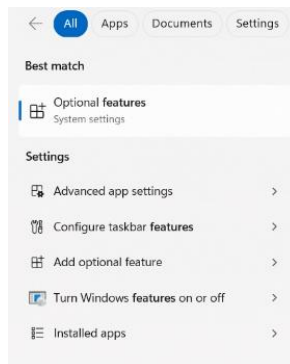
(Catatan: bab ini ditulis berdasarkan bab dengan judul yang sama pada buku Mengenal dan Menggunakan Bahasa Pemrograman Singkong)

Untuk melakukan pengembangan aplikasi web dengan bahasa Singkong, kita membutuhkan sebuah HTTP server (web server) yang mendukung CGI. Di dalam panduan ini, fokus kita adalah menggunakan HTTP server yang telah disertakan oleh sistem operasi yang digunakan, apabila ada. Untuk konfigurasi terkait CGI, kita akan menggunakan konfigurasi default, apabila memang tersedia. User dengan hak administrasi sistem akan diperlukan. Pastikanlah Java telah terinstal.

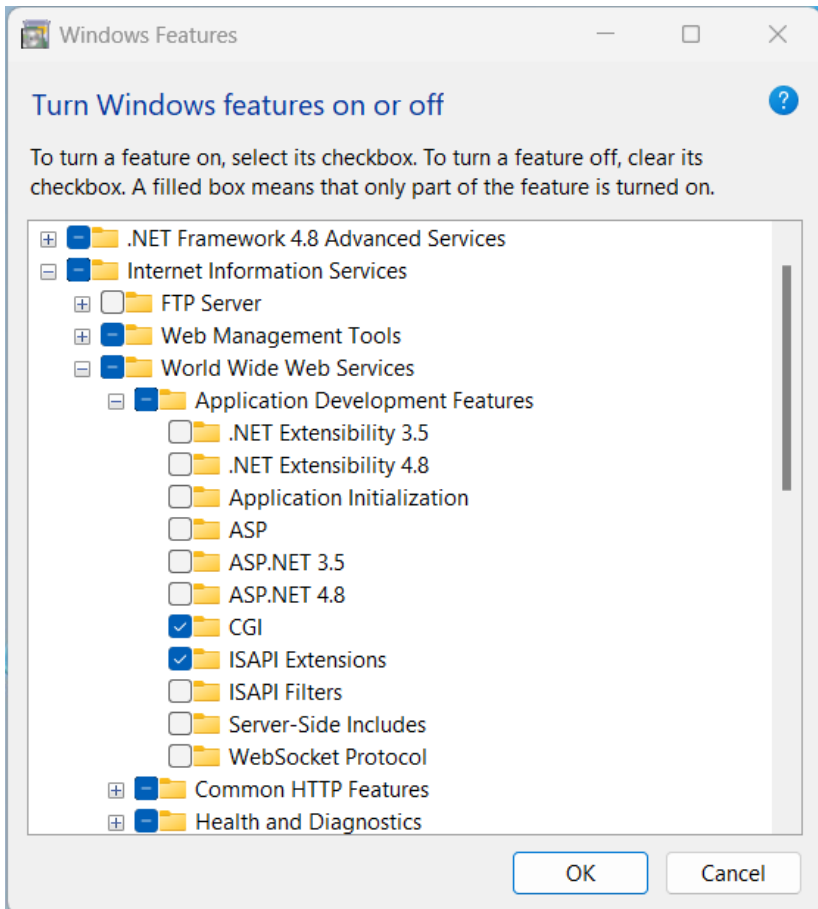
Windows

Untuk Windows, kita akan menggunakan IIS (Internet Information Services), yang merupakan HTTP server bawaan. Panduan ini ditulis berdasarkan Windows 11. Sesuaikanlah apabila diperlukan. Lakukanlah langkah-langkah berikut untuk mengaktifkan IIS.

Pada pencarian di taskbar, ketikkanlah features. Pada hasil pencarian yang tampil, pilihlah Turn Windows features on or off. Anda mungkin perlu melakukan otentikasi sebagai user dengan hak administrasi sistem.



Dialog Windows Features akan ditampilkan. Aktifkanlah IIS dengan klik pada Internet Information Services. Kemudian, kliklah icon tambah pada fitur tersebut untuk menampilkan lebih detail. Tampilkanlah lebih detail World Wide Web Services, lalu pada Application Development Features. Aktifkanlah CGI dan ISAPI Extensions.



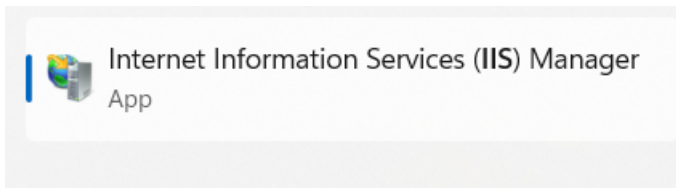
Setelah itu, kliklah tombol OK. Tunggulah sampai fitur ini selesai diaktifkan.

Applying changes

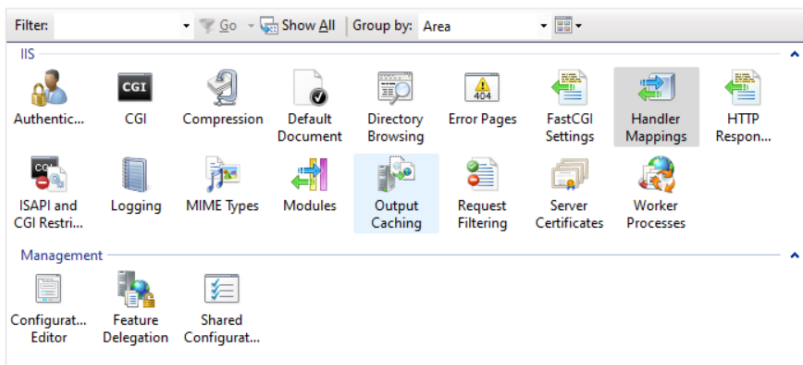


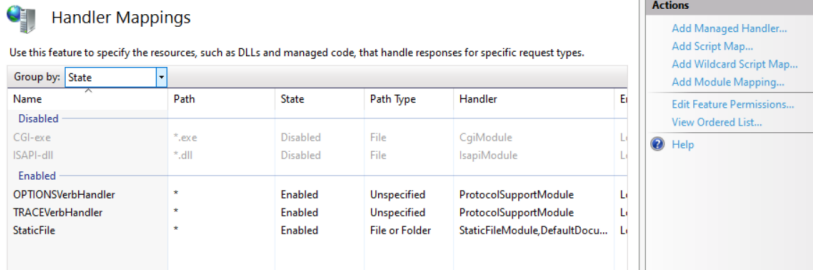
Sampai di sini, IIS telah diaktifkan. Namun, kita perlu melakukan konfigurasi.

Pada pencarian di taskbar, ketikkanlah iis. Pada hasil pencarian yang tampil, pilihlah Internet Information Services (IIS) Manager. Apabila user yang digunakan tidak memiliki hak administrasi sistem, klik kananlah pada item tersebut dan pilihlah Run as administrator.

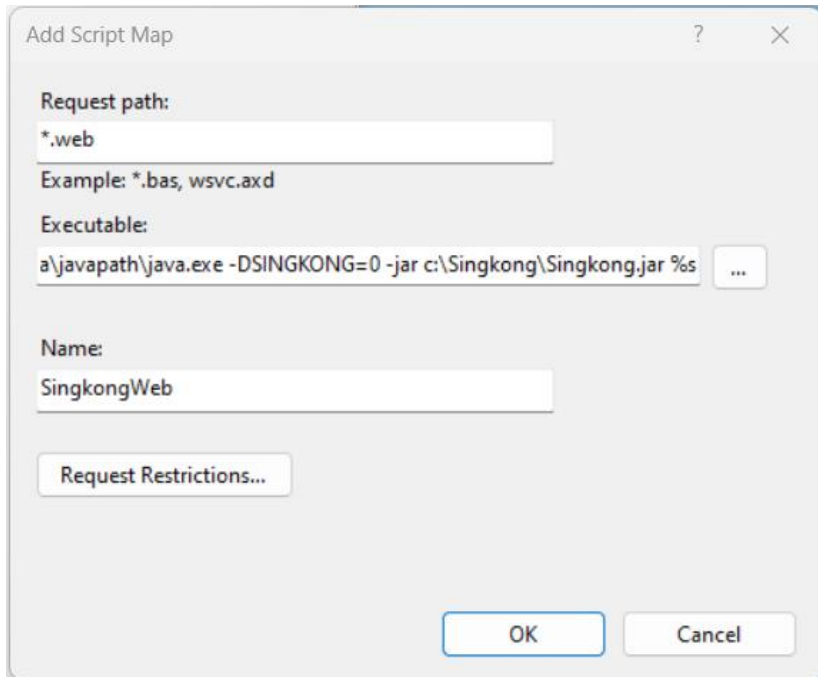


Pada aplikasi yang tampil, jalankanlah Handler Mappings dari Features View.





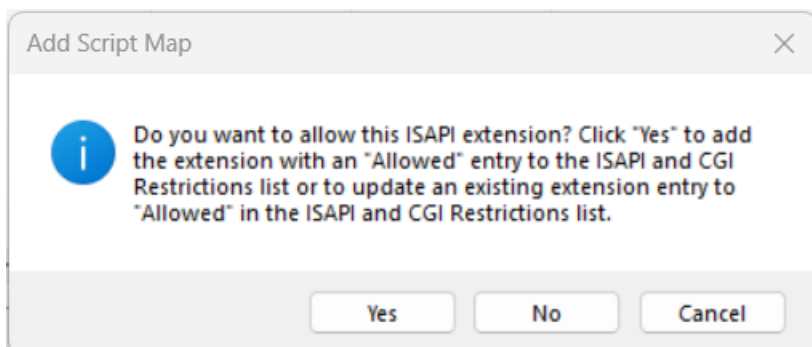
Kemudian, kliklah pada action Add Script Map.... Pada dialog yang tampil:



- Request path: isikanlah *.web.
 - o Catatan: dalam hal ini, file .web (berisi kode Singkong) akan ditempatkan pada \inetpub\wwwroot pada system drive (misal c:)
- Executable: isikanlah <path ke java.exe> -DSINGKONG=0 -jar <path ke Singkong.jar> %s
 - o Contoh:


```
C:\ProgramData\Oracle\Java\javapath\java.exe
-DSINGKONG=0 -jar c:\singkong\Singkong.jar %s
```
 - o Catatan:
 - Kita dapat menggunakan tombol browse ... untuk mencari java.exe
 - Pada cmd, kita dapat mencari java.exe dengan: where.exe java.exe
 - Singkong.jar dapat ditempatkan sesuai preferensi Anda. Sebagai contoh, c:\singkong\Singkong.jar
- Name: isikanlah SingkongWeb

Klik OK untuk menutup dialog. Sebuah dialog konfirmasi untuk mengijinkan ISAPI extension akan ditampilkan. Jawablah Yes.



Kembali pada Handler Mappings, pastikanlah baris SingkongWeb berada pada bagian enabled.



Handler Mappings

Use this feature to specify the resources, such as DLLs and managed code, that handle responses for specific request types.

Name	Path	State	Path Type	Handler
Disabled				
CGI-exe	*.exe	Disabled	File	CgiModule
ISAPI-dll	*.dll	Disabled	File	IsapiModule
Enabled				
OPTIONSVerbHandler	*	Enabled	Unspecified	ProtocolSupportModule
StaticFile	*	Enabled	File or Folder	StaticFileModule,DefaultDocu...
TRACEVerbHandler	*	Enabled	Unspecified	ProtocolSupportModule
SingkongWeb	*.web	Enabled	File	CgiModule

Sampai di sini, konfigurasi pun selesai. Untuk menguji, buatlah file test.web dengan isi berikut:

```
cgi_header()  
print("Test")
```

Kemudian, kopikanlah atau simpanlah file tersebut di dalam direktori wwwroot yang disebutkan sebelumnya.

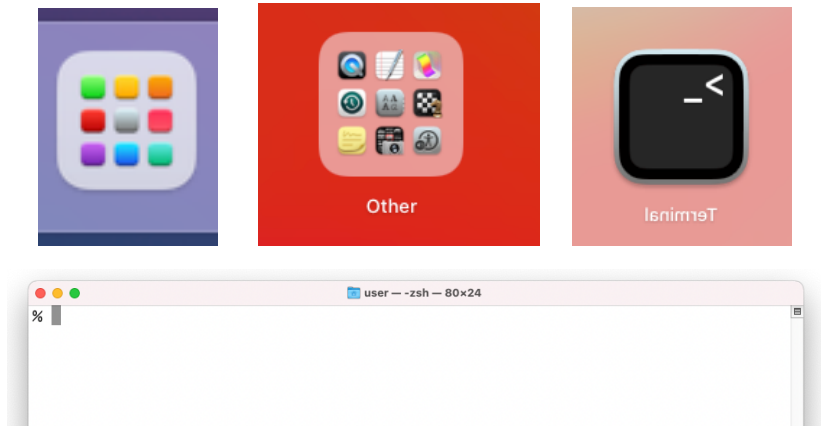
Jalankan web browser dan ketikkanlah alamat berikut:
<http://localhost/test.web>

Apabila konfigurasi telah sesuai, maka tulisan Test akan tampil pada web browser.

macOS

Untuk macOS, kita akan menggunakan Apache HTTP Server bawaan. Panduan ini ditulis berdasarkan macOS 11. Sesuaikanlah apabila diperlukan. Lakukanlah langkah-langkah berikut untuk melakukan konfigurasi.

Pada Dock, bukanlah Launchpad. Pada daftar aplikasi yang tampil, bukanlah folder Other, dan jalankanlah Terminal.



Kita akan bekerja lewat command line pada Terminal. Untuk administrasi sistem, kita akan menggunakan sudo, dengan asumsi user yang digunakan memiliki hak. Editor yang digunakan dalam panduan ini adalah vim. Apabila lebih nyaman bekerja dengan editor lain, misal nano, sesuaikanlah perintah yang dijalankan.

Kita akan mulai dengan mengedit file konfigurasi Apache HTTP Server, yaitu `/etc/apache2/httpd.conf`. Berikanlah perintah berikut dan isikanlah password user apabila diminta.

```
% sudo vim /etc/apache2/httpd.conf
```

Secara default, multi-processing module yang diaktifkan adalah prefork. Apabila dilakukan pencarian terhadap prefork, kita bisa melihat bahwa baris berikut tidak dikomentari (tidak diawali dengan #):

```
LoadModule mpm_prefork_module libexec/apache2/  
mod_mpm_prefork.so
```

Carilah baris `cgi_module` pada `prefork`, dan pastikanlah baris `cgi_module` tidak dikomentari, seperti berikut:

```
<IfModule mpm_prefork_module>
    LoadModule cgi_module libexec/apache2/
mod_cgi.so </IfModule>
```

Modul lain yang perlu diload adalah `actions`:

```
LoadModule actions_module libexec/apache2/
mod_actions.so
```

Kemudian, pastikanlah konfigurasi direktori `/Library/WebServer/CGI-Executables` memiliki opsi untuk menjalankan CGI, seperti berikut ini:

```
<Directory "/Library/WebServer/CGI-Executables">
    AllowOverride None
    Options ExecCGI
    Require all granted
</Directory>
```

Agar file `.web` (berisi kode Singkong) dapat dijalankan, kita perlu menambahkan action dan handler sebagai berikut, pada akhir file konfigurasi:

```
AddHandler SingkongWeb .web
Action SingkongWeb "/cgi-bin/singkongweb.cgi"
```

Simpanlah file dan keluarlah dari editor. Kemudian, restartlah Apache HTTP Server dengan perintah:

```
% sudo apachectl restart
```

Selanjutnya, kita perlu membuat sebuah file dengan nama `singkongweb.cgi`, yang disimpan dalam direktori `/Library/WebServer/CGI-Executables`:

```
% sudo vim /Library/WebServer/CGI-Executables/  
singkongweb.cgi
```

Berikut adalah isi filenya:

```
#!/bin/bash  
  
if [ -z "$PATH_TRANSLATED" ];  
then  
    printf "Status: 404 Not Found\n"  
    printf "Content-type: text/plain\n\n"  
    printf "not found\n"  
else  
    java -DSINGKONG=0 -jar /opt/Singkong.jar  
"$PATH_TRANSLATED"  
fi
```

Dalam file tersebut, `Singkong.jar` diasumsikan telah dikopikan ke `/opt`. Sesuaikanlah apabila diperlukan.

Berikanlah hak akses executable pada file tersebut:

```
% sudo chmod +x /Library/WebServer/CGI-Executables/  
singkongweb.cgi
```

Sampai di sini, konfigurasi telah selesai.

Untuk menguji, buatlah file `test.web` dan simpanlah di `/Library/WebServer/Documents/`:

```
% sudo vim /Library/WebServer/Documents/ test.web
```

Berikut adalah isi filenya:

```
cgi_header()  
print("Test")
```

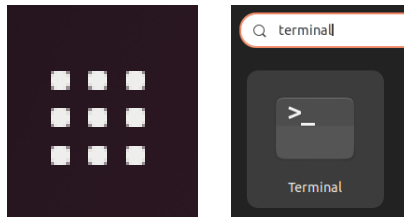
Jalankan web browser dan ketikkan alamat berikut:
`http://localhost/test.web`

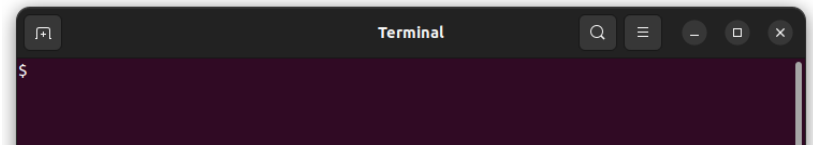
Apabila konfigurasi telah sesuai, maka tulisan `Test` akan tampil pada web browser.

Linux

Distribusi Linux yang digunakan adalah Ubuntu Linux, versi 22.04. Sesuaikanlah apabila diperlukan. Lakukanlah langkah-langkah berikut untuk melakukan instalasi dan konfigurasi Apache HTTP Server. Instalasi akan membutuhkan akses internet.

Bukalah daftar aplikasi dan jalankan Terminal.





Kita akan bekerja lewat command line. Untuk administrasi sistem, kita akan menggunakan sudo, dengan asumsi user yang digunakan memiliki hak. Editor yang digunakan dalam panduan ini adalah nano.

Jalankan perintah berikut untuk melakukan instalasi Apache HTTP Server. Isikanlah password user apabila diminta.

```
$ sudo apt-get update
$ sudo apt-get install apache2
```

Kita perlu mengaktifkan modul actions dan cgi, dengan perintah berikut:

```
$ sudo a2enmod actions cgi
```

Siapkanlah sebuah file konfigurasi tambahan:

```
$ sudo nano /etc/apache2/conf-
available/singkongweb.conf
```

Dengan isi file berikut:

```
AddHandler SingkongWeb .web
Action SingkongWeb "/cgi-bin/singkongweb.cgi"
```

Aktifkanlah file konfigurasi tersebut (pastikan juga untuk serve-cgi-bin):

```
$ sudo a2enconf singkongweb serve-cgi-bin
```

Kemudian, lakukanlah reload:

```
$ sudo systemctl reload apache2
```

Buatlah file `singkongweb.cgi` dengan perintah berikut:

```
$ sudo nano /usr/lib/cgi-bin/singkongweb.cgi
```

Berikut adalah isi filenya:

```
#!/bin/bash

if [ -z "$PATH_TRANSLATED" ];
then
    printf "Status: 404 Not Found\n"
    printf "Content-type: text/plain\n\n"
    printf "not found\n"
else
    java -DSINGKONG=0 -jar /opt/Singkong.jar
"$PATH_TRANSLATED"
fi
```

Dalam file tersebut, `Singkong.jar` diasumsikan telah dikopikan ke `/opt`. Sesuaikanlah apabila diperlukan.

Berikanlah hak akses executable:

```
$ sudo chmod +x /usr/lib/cgi-bin/ singkongweb.cgi
```

Sampai di sini, konfigurasi telah selesai.

Untuk menguji, buatlah file `test.web` dan simpanlah di `/var/www/html`:

```
$ sudo nano /var/www/html/test.web
```

Berikut adalah isi filenya:

```
cgi_header()  
print("Test")
```

Jalankan web browser dan ketikkan alamat berikut:
<http://localhost/test.web>

Apabila konfigurasi telah sesuai, maka tulisan Test akan tampil pada web browser.

Halaman ini sengaja dikosongkan

HTTP dan CGI

Anggap kita memiliki program `hello.web` yang dapat diakses lewat `http://localhost/hello.web`

Cara yang umum untuk menguji program tersebut adalah dengan mengetikkan alamat tersebut di web browser. Apabila HTTP server terinstal dan terkonfigur dengan baik, serta program berfungsi dan akan mencetak output, kita bisa melihat output yang dimaksud di web browser yang digunakan.

Buku ini membahas contoh praktis dan kita tidak membahas detail teknis yang terjadi mulai dari alamat dikunjungi sampai output ditampilkan. Akan tetapi, kita tahu bahwa kita berkomunikasi lewat protokol HTTP (walau kita tidak mengetik `http` pada alamat), dan sebagaimana protokol pada umumnya, terdapat sejumlah aturan.

Untuk bab ini, kita dapat memahami bahwa:

- Terdapat pesan yang dikirimkan dan diterima. Web browser pasti mengirimkan sesuatu dan menerima sesuatu.
- Pesan yang dimaksud, karena kita berbicara lewat protokol HTTP, kita sebut sebagai HTTP message. Baik berupa request (ketika mengirimkan) atau response (ketika server memberikan tanggapan, yang selanjutnya kita terima).
- HTTP message (merujuk pada RFC 2616) terdiri dari start-line, nol atau lebih field header, baris kosong (carriage return dan line feed, juga telah dibahas penggunaannya di buku Contoh dan Penjelasan Bahasa Singkong: Dasar-dasar Aplikasi GUI), dan kemungkinan adanya message body.

Mari kita fokuskan pada field header terlebih dahulu. Program yang kita buat mungkin akan menampilkan form HTML. Atau, konten teks tanpa pemformatan. Atau, data JSON (yang juga kita bahas dalam

buku ini). Atau, data biner dalam format PNG. Untuk tipe media tersebut, kita perlu memberikan nama field header Content-Type, dengan nilai berupa tipe media dimaksud. Antara nama dan nilai, dipisahkan sebuah karakter : (colon). Contoh tipe adalah “text/html”, “text/plain”, “application/json”, “image/png”, dan lainnya.

Program yang kita buat mungkin tidak menghasilkan konten seperti dibahas sebelumnya. Sebagai gantinya, meminta agar dilakukan redirect ke lokasi lain. Untuk itu, kita menggunakan field header Location.

Selain itu, masih terdapat puluhan field lain. Lalu terdapat header untuk kebutuhan lain, misal HTTP State Management (RFC 6265) pada response. Kita tidak akan gunakan secara eksplisit dalam buku ini (fungsionalitas disediakan lewat modul bawaan “web”).

Setelah header, kita sisipkan sebuah baris kosong berupa carriage return dan line feed. Walaupun, dalam berbagai contoh kode, kadang kita menjumpai hanya line feed digunakan.

Setelah itu, barulah, sesuai tipe konten, kita menghasilkan message body.

Sekarang, mari kita bahas sedikit pengantar tentang CGI (Common Gateway Interface versi 1.1, RFC 3875).

Pada implementasi awal World Wide Web, apabila kita melakukan request pada `http://localhost/hello.web` seperti dibahas sebelumnya, kita meminta file `hello.web` yang tersimpan pada file sistem. Bisa kita lihat, ini bersifat statik, karena kita mendapatkan isi file apa adanya.

Sekarang, bayangkanlah kalau apa yang kita request merupakan hasil dari program tertentu yang dijalankan, yang dapat melakukan kalkulasi, melakukan query pada sistem database, dan hal-hal keren

lain, sesuai environment ketika program berjalan. Kini, sifatnya menjadi dinamis dan menjadi dasar dari aplikasi web yang kita kenal.

Merujuk pada `http://localhost/hello.web`, secara dinamis, ini artinya, Java virtual machine dijalankan, yang kemudian menjalankan interpreter Singkong (`Singkong.jar`), yang kemudian melakukan interpretasi terhadap isi file `hello.web`, dan barulah output dihasilkan.

Dari sejak HTTP server menerima request sampai output dari program dihasilkan, kita tentunya membutuhkan cara kerja tertentu, yang dalam pembahasan kita, berupa Common Gateway Interface. CGI, yang digunakan pada World Wide Web sejak 1993 memungkinkan HTTP server dan program CGI untuk menangani request.

Singkong menyediakan sejumlah fungsi bawaan dan modul web untuk bekerja dengan CGI.

Catatan: Dari ilustrasi cara kerja sebelumnya, dengan CGI, setiap request akan memicu sebuah program dijalankan. Artinya, secara umum, bisa jadi dibutuhkan virtual machine, interpreter, dan lain sebagainya ikut dijalankan. Konsekuensinya, dibutuhkan sumber daya komputasi tertentu pada sistem operasi. Terdapat sejumlah alternatif CGI, namun setidaknya sampai buku ini ditulis, Singkong hanya menyediakan CGI.

Pengantar kita sudah selesai. Mari kita bahas beberapa contoh.

Apa yang perlu kita sediakan dalam program kita hanyalah urusan mencetak ke standard output dengan fungsi `print` misalnya. Yang penting, sesuai protokol HTTP.

Jadi, cukup fungsi `print` saja? Baiklah. Mungkin bersama fungsi `env` dan `stdin`. Lalu, kenapa di referensi dan buku Singkong lainnya, `env`

dan stdin tidak pernah dicontohkan untuk pemrograman aplikasi web?

Hal ini karena Singkong menyediakan sejumlah fungsi bantu untuk mempermudah, yang mana disarankan penggunaannya. Dalam buku ini, yang akan kita bahas adalah fungsi-fungsi tersebut.

Dalam berbagai contoh aplikasi web dengan Singkong, kita mungkin menjumpai kode berikut:

```
cgi_header()
```

Sebenarnya, cukup baris tersebut saja, dan kita sudah membuat aplikasi web dengan bahasa Singkong. Hanya saja, tidak ada output apapun di browser.

Untuk menghasilkan output, kita dapat menggunakan fungsi `cgi_contents` atau dengan `print/println`. Dengan demikian, untuk mencetak Hello misalnya, kita cukup menambahkan baris berikut:

```
println("Hello")
```

Alih-alih mencetak ke terminal atau menampilkan message box, kita mendapatkan Hello tersebut di web browser (atau, apapun http client yang digunakan).

Tapi, Anda mungkin bertanya. Bukankah `println` tersebut, apabila GUI tersedia, akan menampilkan message box? Benar. Dan, oleh karena itu, di konfigurasi web server, umumnya kita menambahkan `-DSINGKONG=0` untuk memastikan Singkong berjalan pada mode teks, bahkan ketika GUI tersedia.

Sampai di sini, kita sebenarnya bisa mengakhiri pembahasan. Tapi, rasanya kurang seru kalau kita tidak membahas kemungkinan adanya error.

Bagaimana kalau kita lupa mengirimkan header? Misal lupa memanggil fungsi `cgi_header`? Mari kita lihat:



Pada komputer dengan Internet Information Services yang penulis gunakan, kita bisa melihat bahwa terjadi Bad Gateway, di mana message yang dikirimkan, yang harusnya berupa header (setelah start line) adalah Hello. Ini bukanlah field header yang valid.

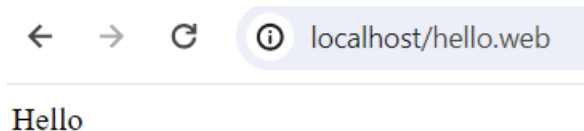
Kode error 502 ini merupakan kategori error di sisi server (5xx), yang menandakan bahwa server menerima response yang tidak valid dari upstream (dalam hal ini, program kita).

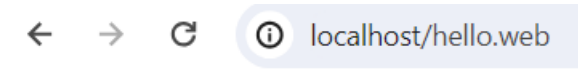
Apa yang dilakukan oleh `cgi_header` ketika kita tidak memberikan argumen apapun, seperti contoh sebelumnya? Fungsi akan mengirimkan header `Content-Type: text/html`.

Bagaimana kalau kita ingin mengirimkan header lain, misal konten kita berupa plain text? Fungsi menerima sebuah argumen, yaitu HASH, dengan key berupa field header dan value berupa nilai untuk field tersebut. Sebagai contoh:

```
cgi_header({"Content-Type": "text/plain"})
```

Pada web browser, kita *mungkin* bisa membedakan antara konten berupa HTML dan plain text, seperti pada gambar-gambar berikut:





Hello

Bagaimana kalau kita mengirimkan header lain, misal Location?

```
cgi_header({"Location": "helloworld.web"})
```

Redireksi ke helloworld.web akan dilakukan. Apabila tidak ditemukan, maka client error 4xx, dalam hal ini 404 Not Found yang akan terjadi.

Kita bisa melihat bahwa dengan mencetak nilai-nilai tertentu ke standard output sesuai aturan HTTP, kita bisa mendapatkan hasil yang berbeda sama sekali.

Kita tidak harus menggunakan fungsi `cgi_header` tersebut. Di berbagai contoh program CGI, Anda mungkin melihat bahwa kita dapat menggunakan `print` saja, seperti contoh berikut:

```
println("Content-Type: text/plain")
println("Hello")
```

Apabila Anda mencoba kode tersebut, tentu saja error bad gateway akan kembali terjadi. Kita perlu menambahkan baris kosong setelah header. Per aturan HTTP, ini adalah carriage return dan line feed. Maka, kita bisa tambahkan dengan fungsi `cr` dan `lf` (atau `crlf`):

```
println("Content-Type: text/plain")
print(cr() + lf())
println("Hello")
```

Atau, pada RFC 2616, pada bagian 19.3 Tolerant Applications, kita bisa menemukan:

```
The line terminator for message-header fields is the sequence CRLF. However, we recommend that applications, when parsing such headers, recognize a single LF as a line terminator and ignore the leading CR.
```

Maka, contoh berikut dapat digunakan:

```
println("Content-Type: text/plain")
print(lf())
println("Hello")
```

Atau, digabung dengan println sebelumnya, apabila dimaksudkan mengakhiri header:

```
println("Content-Type: text/plain" + lf())
println("Hello")
```

Di buku ini, kita akan tetap menggunakan fungsi `cgi_header`, diantaranya bahwa kita dapat mengirimkan beberapa header dengan lebih nyaman, seperti cuplikan kode berikut (yang selengkapnya akan kita bahas pada bagian session):

```
    cgi_header (
        header_session(sess) +
        header_location("home.web")
    )
```

Apabila pembahasan di bab ini terkesan bertele-tele, tujuan kita diantaranya adalah kita memahami lebih dalam apa yang terjadi, sehingga ketika terjadi error misalnya, kita dapat melakukan troubleshooting yang lebih tepat.

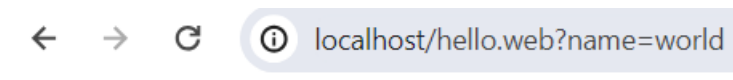
Catatan: Dengan membaca environment variable tertentu, yang bisa didapatkan dengan fungsi env, kita bisa mendapatkan informasi seperti contoh berikut:

```
var e = env()
cgi_header()
print(
    "<!DOCTYPE html>
    <html lang='en'>
        <head>
            <title>Env</title>
        </head>
        <body>
            Remote address: " + e["REMOTE_ADDR"]
+ "<br>
            HTTP user agent: " +
e["HTTP_USER_AGENT"] + "
        </body>
    </html>"
)
```

Request Method GET dan Query String

Pada contoh `hello.web` sebelumnya, kita melakukan request dengan method GET lewat web browser. Di buku ini, diasumsikan kita mengetikkan alamat <http://localhost/hello.web> tersebut, atau dengan klik link apabila membaca buku ini lewat format PDF. Di kesempatan lain, kita mungkin mengakses sebuah aplikasi web lewat scan pada kode QR.

Fokus kita saat ini bukan pada konten yang didapatkan sebagai respon dari web server. Kita lebih tertarik pada `hello.web` itu sendiri. Mari kita tambahkan parameter sehingga menjadi `hello.web?name=world`:



Hello

Bisa kita lihat, tidak ada bedanya dengan ketika kita hanya mengetikkan `hello.web` saja. Kenapa? Karena memang tidak kita tangani dalam kode `hello.web` berikut:

```
cgi_header()  
println("Hello")
```

Anda mungkin bertanya. Untuk apa hal yang sudah sejelas itu dibahas?

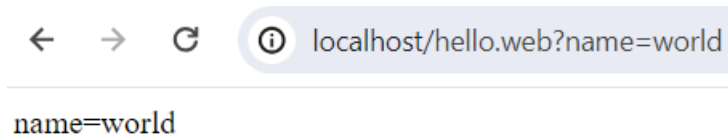
Di RFC 3986 tentang Uniform Resource Identifier (URI): Generic Syntax, kita dapat membaca bahwa apa yang kita berikan sebelumnya disebut sebagai query, yang dimulai setelah tanda tanya pertama sampai akhir, atau sebelum penanda fragment `#` apabila ada.

Ketika request tersebut dilakukan, per RFC 3875 (CGI 1.1), sejumlah meta-variable dilewatkan oleh web server ke program CGI. Yang menjadi perhatian kita sekarang adalah QUERY_STRING.

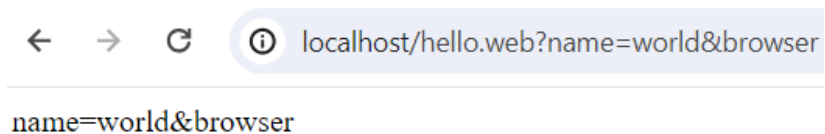
Mari kita sesuaikan program hello.web kita menjadi demikian:

```
cgi_header ()  
  
println (env () ["QUERY_STRING"])
```

Kemudian, kunjungilah kembali alamat sebelumnya, yaitu <http://localhost/hello.web?name=world>



Bisa kita lihat, kita mendapatkan query string tersebut. Walaupun umum diberikan dalam key=value, ini tidaklah wajib. Kita bisa saja berikan seperti:



Sekarang, untuk mendapatkan world dari name=world, apakah kita harus melakukan parsing secara manual? Tentu saja tidak. Kita bisa gunakan fungsi `cgi_get`, yang mendapatkan QUERY_STRING sebagai sebuah HASH. Kembali kita sesuaikan program hello.web kita menjadi:

```
cgi_header ()  
  
println (cgi_get ())
```

← → ↻ ⓘ localhost/hello.web?name=world&browser

```
{"name": "world", "browser": ""}
```

Contoh penanganan query di Singkong:

QUERY_STRING	HASH
a	{"a": ""}
a&b	{"a": "", "b": ""}
a=1	{"a": "1"}
a=1&a=2	{"a": ["1", "2"]}
a=1&a=2&a&b=1&b=2&b	{"a": ["1", "2", ""], "b": ["1", "2", ""]}
a=1&a=2&A&b=1&b=2&b&c	{"a": ["1", "2"], "A": "", "b": ["1", "2", ""], "c": ""}
a=hello+world	{"a": "hello world"}
a=hello%20world!	{"a": "hello world!"}

Dari contoh tersebut, bisa kita lihat bahwa proses decoding dilakukan otomatis (hello+world dan hello%20world menjadi hello world), serta key dengan nama sama / diberikan berulang akan otomatis menjadi ARRAY. Apabila diberikan hanya nama key saja, akan diberikan nilai berupa STRING kosong.

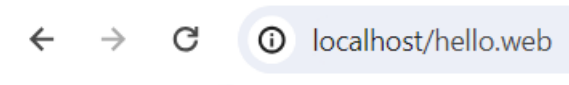
Dengan demikian, untuk mendapatkan name, kita bisa sesuaikan kembali program hello.web kita:

```
cgi_header()  
println("Hello " + cgi_get()["name"])
```

← → ↻ ⓘ localhost/hello.web?name=world

Hello world

Sampai di sini, Anda mungkin berpendapat, kode tersebut tidak menangani ketika name tidak diberikan:



Hello null

Benar sekali. Tidak ada error yang terjadi, karena apabila key tidak tersedia di HASH, null akan dikembalikan. Akan tetapi, akan lebih baik apabila kita tampilkan name hanya kalau diberikan. Sesuaikanlah hello.web menjadi:

```
cgi_header()
var g = cgi_get()
var n = g["name"]
var m = "Hello "
if (n != null) {
    var m = m + n
}
println(m)
```

Penjelasan:

- **Diwarnai merah:** kita dapatkan query string ke dalam sebuah HASH, sehingga bisa digunakan kembali apabila diperlukan nantinya.
- **Diwarnai biru:** perhatikanlah bahwa n mungkin bernilai null di sini, apabila name tidak diberikan. Perhatikanlah bahwa n mungkin bernilai STRING kosong, yang dalam contoh ini, tidak kita tangani secara khusus.
- **Diwarnai hijau:** nilai default yang akan ditampilkan adalah Hello, ada atau tidak name dilewatkan nantinya.

- **Diwarnai orange:** hanya apabila n tidak null, artinya name diberikan, kita tambahkan ke m. Apabila n bernilai STRING kosong, kita tetap tambahkan.

Bagaimana kalau Hello tersebut ditujukan kepada sejumlah nama sekaligus? Misal:

```
← → ↻ ⓘ localhost/hello.web?name=A&name=B&name=C
```

Hello ["A", "B", "C"]

Bisa kita lihat, hello.web tadi masih bisa diperbaiki:


```
cgi_header()
var g = cgi_get()
var n = g["name"]
var m = "Hello "
if (n != null) {
    if (is(n, "ARRAY")) {
        var m = m + join(", ", n)
    } else {
        var m = m + n
    }
}
println(m)
```

```
← → ↻ ⓘ localhost/hello.web?name=A&name=B&name=C
```

Hello A, B, C

Di dalam contoh tersebut, kita memroses tersendiri apabila name diberikan berkali-kali.

Akan tetapi, tentunya, karena key bisa diberikan tanpa value, maka kita belum menangani contoh berikut:



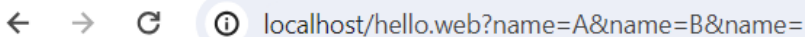
← → ↻ ⓘ localhost/hello.web?name=A&name=&name=

Hello A, ,

Kita sesuaikan sedikit lagi, sehingga hello.web menjadi:

```
cgi_header()
var g = cgi_get()
var n = g["name"]
var m = "Hello "
if (n != null) {
    if (is(n, "ARRAY")) {
        var nn = []
        each(n, fn(e, i) {
            var ee = trim(e)
            if (!empty(ee)) {
                nn + ee
            }
        })
        var m = m + join(", ", nn)
    } else {
        var m = m + n
    }
}
println(m)
```

Kita hanya proses ketika key memiliki value tertentu:



← → ↻ ⓘ localhost/hello.web?name=A&name=B&name=

Hello A, B

Sampai di sini pembahasan kita. Terima kasih.

Form: GET

Di contoh sebelumnya, kita melihat bagaimana query diberikan dengan mengetikkan langsung. Akan tetapi, rasanya ini tidak umum ketika kita menggunakan aplikasi web. Alih-alih mengetikkan `hello.web?name=world` seperti contoh sebelumnya, lebih umum pengguna disajikan sebuah input berupa teks dan tombol.

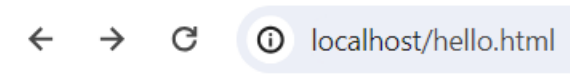
Input teks dan tombol tersebut dapat menjadi bagian dari sebuah form HTML. Di bab ini, kita akan membahas beberapa contoh dasar penanganan form. Karena buku ini tidak membahas HTML secara spesifik, contoh form kita akan cukup sederhana.

Kita akan mulai dengan membuat form untuk contoh sebelumnya. Kita sudah memiliki `hello.web` yang kelihatannya berfungsi baik, karena dapat menangani ada atau tidak name diberikan, dan apabila diberikan, dapat berupa nilai tunggal atau sejumlah name.

Buatlah `hello.html` dengan contoh isi berikut:

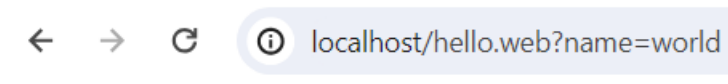
```
<!DOCTYPE html>
<html lang='en'>
  <head>
    <title>Hello</title>
  </head>
  <body>
    <form action='hello.web'>
      Name: <input type='text' name='name'>
        <input type='submit' value='Hello'>
    </form>
  </body>
</html>
```

Akseslah form tersebut lewat `http://localhost/hello.html`



Name:

Isikanlah world pada input Name dan kliklah tombol Hello. Tentu saja, kita akan mendapatkan Hello world seperti contoh sebelumnya:



Hello world

Perhatikanlah `hello.web?name=world` tersebut. Sama dengan ketika kita mengetikkan langsung di contoh sebelumnya. Kita sudah membuat sebuah form dengan action merujuk pada `hello.web` dan method berupa GET (default, ketika action tidak ditentukan secara eksplisit).

Kita tahu bahwa `hello.web` kita dapat menangani lebih. Form sebelumnya hanya memungkinkan kita untuk mengucapkan Hello dengan input name tunggal. Mari kita lengkapi form `hello.html` sebelumnya (baris baru diformat tebal), menjadi:

```
<!DOCTYPE html>
<html lang='en'>
  <head>
    <title>Hello</title>
  </head>
  <body>
    <form action='hello.web'>
      Name: <input type='text' name='name'>
      Name: <input type='text' name='name'>
      Name: <input type='text' name='name'>
      <input type='submit' value='Hello'>
    </form>
```

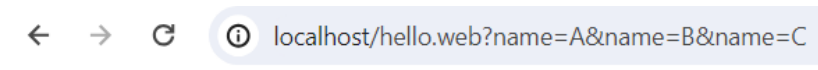
```
</body>  
</html>
```

Benar sekali. Kita hanya menambah input, dengan name yang sama persis (name='name'). Kunjungilah kembali hello.html dan isikanlah misal A, B, dan C pada masing-masing name, kemudian kliklah tombol Hello:



A screenshot of a web browser window. The address bar shows 'localhost/hello.html'. Below the address bar, there is a form with three input fields, each preceded by the text 'Name:'. The first field contains 'A', the second contains 'B', and the third contains 'C'. To the right of the third field is a button labeled 'Hello'.

Kita akan mendapatkan tampilan yang familiar:



A screenshot of a web browser window. The address bar shows 'localhost/hello.web?name=A&name=B&name=C'. The browser has navigation buttons (back, forward, refresh) and an information icon to the left of the address bar.

Hello A, B, C

Perhatikanlah bahwa tidak ada perbedaan dalam cara kita menuliskan nama variabel ketika dimaksudkan sebagai input array. Fungsi `cgi_get` secara otomatis akan membuat ARRAY apabila perlu.

Pembahasan untuk bab ini selesai. Mari kita bahas form dengan method POST.

Halaman ini sengaja dikosongkan

Form: POST

Apabila kita perhatikan, dengan method GET, baik dengan form atau diketikkan sendiri, query akan menambah panjang alamat (URI). RFC 9110 tentang HTTP Semantics, pada URI References, merekomendasikan panjang URI adalah 8000 oktet. Sementara, RFC 2616 tentang HTTP/1.1, pada General Syntax, memberikan catatan terkait client versi lama yang mungkin tidak mendukung panjang lebih dari 255 byte. Browser juga mungkin memiliki batasan yang dapat berbeda.

Bagaimana kalau kita perlu mengirimkan data dengan ukuran yang jauh lebih besar, puluhan MB misalnya? Kita dapat menggunakan method POST, dimana panjangnya tidak dibatasi secara spesifik.

Pada form HTML, untuk mengubah dari GET pada contoh sebelumnya menjadi POST, kita cukup menambahkan `method='POST'` pada form, seperti contoh berikut (kita simpan pada file `post.html`; penyesuaian diformat tebal):

```
<!DOCTYPE html>
<html lang='en'>
  <head>
    <title>Hello</title>
  </head>
  <body>
    <form action='post.web' method=POST'>
      Name: <input type='text' name='name'>
      Name: <input type='text' name='name'>
      Name: <input type='text' name='name'>
      <input type='submit' value='Hello'>
    </form>
  </body>
</html>
```

Kita mengubah action form ke post.web. Apabila tetap ditangani dengan hello.web, ketika post.html dikunjungi dan tombol Hello di klik, isi form tidak bisa didapatkan oleh hello.web, seperti contoh berikut:

The image shows two browser screenshots. The top screenshot shows a browser at localhost/post.html with a form containing three input fields labeled 'Name: A', 'Name: B', and 'Name: C', and a 'Hello' button. The bottom screenshot shows the browser at localhost/hello.web displaying the word 'Hello'.

Hello

Hal ini karena hello.web menggunakan fungsi `cgi_get`, dan form kita menggunakan method POST.

Mari kita lihat isi post.web (perbedaan dengan hello.web diformat tebal):

```
cgi_header()  
var g = cgi_post()  
var n = g["name"]  
var m = "Hello "  
if (n != null) {  
    if (is(n, "ARRAY")) {  
        var nn = []  
        each(n, fn(e, i) {  
            var ee = trim(e)  
            if (!empty(ee)) {  
                nn + ee  
            }  
        })  
        var m = m + join(", ", nn)  
    } else {  
        var m = m + n  
    }  
}  
println(m)
```


Bedanya hanya pada `cgi_get` yang diganti menjadi `cgi_post`.
Selainnya, sama persis. Dan, kita mendapatkan isi form:

A screenshot of a web browser's address bar. It features navigation icons (back, forward, refresh) on the left, an information icon, and the URL 'localhost/post.web' in a light blue background.

Hello A, B, C

Apabila diperhatikan, tidak ada query string pada `post.web`. Hal ini disebabkan karena pada `POST`, kita tidak menggunakan `QUERY_STRING` seperti sebelumnya. Isi form dilewatkan pada standard input (`stdin`) dan fungsi `cgi_post` membaca dari standard input, sampai sepanjang environment variable `CONTENT_LENGTH`, apabila diset. Cara penanganan isi form sama seperti pada tabel Contoh penanganan query di Singkong yang dibahas sebelumnya.

Catatan: MIME type yang didukung pada fungsi-fungsi bawaan `cgi_get`, `cgi_post`, dan `cgi_post_hash` adalah `application/x-www-form-urlencoded`. Pada saat buku ini ditulis, Singkong belum mendukung `multipart/form-data`.

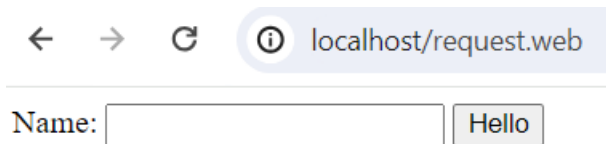
Halaman ini sengaja dikosongkan

Memeriksa Request Method

Kita akan mulai bab ini dengan pertanyaan: kenapa kita membuat hello.html dan post.html apabila kita dapat print semua konten tersebut dari sebuah file .web? Perhatikanlah contoh request.web berikut (yang isinya akan kita ubah nanti):

```
cgi_header()
print ("
  <!DOCTYPE html>
  <html lang='en'>
    <head>
      <title>Hello</title>
    </head>
    <body>
      <form method='POST'>
        Name: <input type='text' name='name'>
          <input type='submit' value='Hello'>
      </form>
    </body>
  </html>
")
```

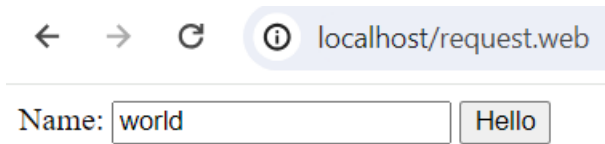
Ketika dikunjungi, kita akan mendapatkan form yang familiar:



← → ↻ ⓘ localhost/request.web

Name:

Cobalah isikan name dan klik tombol Hello:



Namun, kita malah kembali mendapatkan form, dengan name tidak diisi.

Yang terjadi adalah:

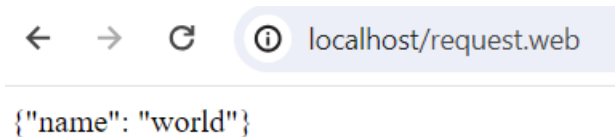
- Kita akan tetap mengunjungi request.web, karena kita tidak spesifik menentukan action pada form, dan form akan ditangani oleh request.web.
- Kita tidak menangani form secara eksplisit ketika metode request adalah POST. Form akan kembali dihasilkan.

Baiklah, mari kita ubah lagi file request.web menjadi berikut (nantinya isinya akan diubah lagi):

```
load_module("web")
if (request_method() == "GET") {
    cgi_header()
    print("
        <!DOCTYPE html>
        <html lang='en'>
        <head>
            <title>Hello</title>
        </head>
        <body>
            <form method='POST'>
                Name: <input type='text' name='name'>
                <input type='submit' value='Hello'>
            </form>
        </body>
    </html>
    ")
} else {
```

```
    cgi_header()
    print(cgi_post())
}
```

Kembalilah mengunjungi request.web. Isikanlah name dan kliklah tombol Hello. Kita akan mendapatkan versi STRING dari HASH yang dikembalikan oleh cgi_post.



Kembali ke pertanyaan di awal bab. Dalam versi terakhir request.web, kita tidak membuat file HTML tersendiri untuk form. Kita hasilkan dari request.web ketika request method adalah GET. Ketika form dengan method POST ditangani, kita kembali tangani dengan request.web yang sama.

Sebelum kita mengubah isi request.web lagi, mari kita cermati hal-hal berikut:

- Yang dirasa langsung adalah jumlah file menjadi lebih sedikit. Hanya ada request.web.
- Walau, file request.web tersebut menjadi lebih kompleks dengan adanya if untuk memeriksa request method. Lalu, kita perlu menggunakan fungsi untuk menghasilkan form (kode HTML), seperti print.
- Kemudian, sebagaimana dibahas pada instalasi dan konfigurasi web server, kita perlu menjalankan Java virtual machine, kemudian interpreter Singkong, dan interpreter Singkong tersebut perlu menjalankan kode-kode dalam request.web. Tentunya butuh sumber daya komputasi

tertentu dibandingkan ketika web server langsung memberikan isi file HTML.

Sekarang, mari kita lihat kembali request.web sebelumnya. Terdapat sejumlah request method lain (misal HEAD, PUT, DELETE), walau untuk bab ini, kita hanya perlu menangani GET dan POST. Bagaimana kalau kita tulis ulang menjadi berikut:

```
load_module("web")
if (request_method() == "GET") {
    cgi_header()
    print("
        <!DOCTYPE html>
        <html lang='en'>
        <head>
            <title>Hello</title>
        </head>
        <body>
            <form method='POST'>
                Name: <input type='text' name='name'>
                <input type='submit' value='Hello'>
            </form>
        </body>
    </html>
    ")
    exit()
}

if (request_method() == "POST") {
    cgi_header()
    print(cgi_post())
    exit()
}
```

Di Singkong, kita tidak mengenal semacam else if. Untuk if, hanya terdapat else. Daripada kita melakukan if lagi dalam blok else, terkadang kita bisa gunakan fungsi exit seperti dicontohkan, agar

program langsung diterminasi. If berikutnya tidak lagi penting setelah blok if yang tepat dikerjakan.

Baik, penulis sudah menduga Anda akan berpendapat demikian: apa guna exit di sini, apabila setelahnya toh kita tidak melakukan apa-apa. Apabila request method adalah GET, kita toh tidak akan masuk dalam if di mana request method adalah POST.

Tentu saja penulis sependapat. Hanya saja, apabila kita terbiasa dengan cara demikian, kadang-kadang kita akan melakukan sesuatu, seperti redireksi ke halaman lain, ketika semua if tidak terpenuhi. Dengan demikian, kita mungkin akan memiliki baris seperti berikut di akhir file:

```
cgi_header({"Location": "login.web"})
```

Di contoh kita, baris tersebut tidak diperlukan. Tapi, `cgi_header` juga akan mencetak ke standard output dan apabila baris tersebut ditambahkan tanpa adanya pemanggilan exit sebelumnya, maka akan ikut tampil dalam konten HTML misalnya. Cobalah untuk menghapus exit dan lihatlah perbedaannya (sambil header Location ditambahkan).

Sebelum mengakhiri pembahasan, apakah Anda mencermati bahwa kita menggunakan module web? Contoh kita membutuhkan fungsi `request_method` dari module tersebut. Di bab berikutnya, kita akan bekerja dengan session, dimana beberapa fungsi dari module web akan digunakan.

Halaman ini sengaja dikosongkan

Bekerja dengan Session

Di contoh form Hello sebelumnya, kita melakukan request dan form ditampilkan. Kemudian, kita isikan name, klik tombol Hello, dan Hello untuk setiap name yang diisikan akan ditampilkan. Ini adalah request yang terpisah. Kita bahkan bisa langsung melakukan request yang mensimulasikan penekanan tombol tersebut, tanpa harus melewati tahapan form ditampilkan (kita akan bahas ini di bab lain). Tidak ada state yang dimaintain secara otomatis diantara request-request tersebut. Oleh karena itu, HTTP disebutkan sebagai protokol yang stateless.

Sekarang, bayangkan ketika ada sejumlah halaman yang hanya bisa dikunjungi ketika user telah melakukan otentikasi. Program di sisi server harus mengetahui bahwa terdapat konsep sesi untuk user tersebut, sehingga (A) halaman-halaman tersebut tersedia, dan (B) user tidak selalu diminta untuk memasukkan username dan password. Bayangkan betapa merepotkannya apabila di setiap halaman tersebut, user harus selalu mengisi password.

Kita butuh penanda bukan? Cara yang umum digunakan adalah menggunakan HTTP cookie, yang merupakan data yang dikirimkan oleh server, dan diasumsikan disimpan oleh browser. Ketika browser melakukan request lain (yang tentunya merupakan request terpisah), data tersebut ikut dikirimkan kembali. Itulah penanda yang kita maksudkan. Sekarang, kita bisa menandai sekian request terpisah sebagai sesi tersendiri. Dalam konteks otentikasi, apabila berhasil dilakukan, kita buat sesi baru. Selama masih dalam sesi yang sama, sejumlah halaman dan tindakan diijinkan. Pada akhirnya, sebuah sesi bisa diakhiri.

Walaupun sekilas terlihat sederhana, urusan mengelola state ini kompleks dan RFC 6265 tentang HTTP State Management saja terdiri

dari total 37 halaman. RFC 2616 tentang HTTP/1.1 dari tahun 1999 saja (dengan sejumlah RFC yang lebih baru), terdiri dari total 176 halaman.

Penulis tahu bahwa Anda mungkin akan mengingatkan: bukankah buku ini tentang contoh praktis?

Baiklah, mari kita kembali ke pembahasan tentang session. Kita sudah mempelajari tentang mengirimkan header sejak awal. Membuat cookie juga sama, dengan mengirimkan header Set-Cookie. Ketika browser melakukan request lain, header Cookie dikirimkan.

Module bawaan web yang digunakan di bab sebelumnya datang bersama beberapa fungsi yang terkait penggunaan session. Kita tinggal gunakan, tanpa harus repot mengirimkan header dan memeriksa header yang dikirimkan balik.

Anggap kita punya cara kerja berikut:

- Kita punya empat halaman: `index.web`, `login.web`, `home.web`, dan `logout.web`.
- User boleh mengunjungi secara langsung halaman-halaman tersebut. Kita yang harus menentukan apa yang harus dilakukan ketika halaman-halaman tersebut dikunjungi.
- Halaman `home.web` hanya boleh dikunjungi ketika user sudah melakukan otentikasi. Apabila belum, diarahkan ke `login.web`.
- Halaman `login.web` hanya boleh dikunjungi ketika user belum melakukan otentikasi. Apabila sudah, diarahkan ke `home.web`. Apabila otentikasi berhasil, sesi baru dibuat.
- Halaman `index.web`, apabila dikunjungi, akan memeriksa apakah terdapat sesi user, dan melakukan redireksi sebagaimana diperlukan.

- Halaman logout.web akan menghapus sesi yang ada, kemudian mengarahkan ke login.web.

Terdengar wajar bukan? Mari kita mulai dari index.web. Kodenya tidaklah panjang:

```
load_module("web")
var sess_path = "c:\session"
var sess = session_file_get(sess_path)
if ((sess["user"] != null) & (sess["session"] !=
null)) {
    cgi_header(
        header_location("home.web")
    )
    exit()
}

cgi_header(
    header_location("login.web")
)
```

Di sisi web server, kita perlu menentukan sebuah direktori yang dapat ditulis oleh proses web server. Di komputer yang penulis gunakan dengan web server IIS (Internet Information Services), sebuah direktori c:\session dibuat untuk menampung file-file data session. Penulis cukup membuat direktori tersebut dan tidak ada konfigurasi tambahan yang dilakukan. Sesuaikanlah dengan sistem operasi dan web server yang digunakan.

Kita perlu dapatkan data session tersimpan untuk setiap halaman yang akan bekerja dengan session. Nantinya, kita akan dapatkan sebagai sebuah HASH.

```
var sess_path = "c:\session"
var sess = session_file_get(sess_path)
```

Kemudian, kita cek apakah key user (atau apapun key selain yang di-reserve: session dan timestamp) dan session tidak null. Apabila demikian, maka diasumsikan telah terdapat sesi user dan diarahkan ke home.web dengan header location.

```
if ((sess["user"] != null) & (sess["session"] !=
null)) {
    cgi_header(
        header_location("home.web")
    )
    exit()
}
```

Perhatikanlah setelah mengirimkan header, kita terminasi program. Andaikata pemeriksaan sesi ini gagal, pada akhir file, kita minta user untuk login:

```
cgi_header(
    header_location("login.web")
)
```

Semua dasar yang diperlukan telah kita bahas sebelumnya.

Begitupun juga dengan login.web. Kodenya agak panjang karena kita buat form dan tangani di file yang sama. Kita memeriksa request method seperti contoh sebelumnya. Berikut adalah isi filenya:

```
load_module("web")
var sess_path = "c:\session"
var sess = session_file_get(sess_path)
if ((sess["user"] != null) & (sess["session"] !=
null)) {
    cgi_header(
        header_location("home.web")
    )
    exit()
}
```

```

if (request_method() == "GET") {
    var g = cgi_get()
    var error = ""
    if (g["error"] == "auth") {
        var error = "Authentication failed"
    }

    cgi_header()
    print("
        <!DOCTYPE html>
        <html lang='en'>
            <head>
                <title>Login</title>
            </head>
            <body>
                "
                + error +
                "
                <form action='login.web'
method='post'>
                    Username:
                    <input type='text' name='u'>
                    Password:
                    <input type='password'
name='p'>
                    <input type='submit'
value='login'>
                </form>
            </body>
        </html>
    ")
    exit()
}

if (request_method() == "POST") {
    var form = cgi_post()
    var u = form["u"]
    var p = form["p"]

```

```

if (u == "admin" & p == "admin") {
    var sess = session_new()
    set(sess, "user", u)
    session_file_set(sess_path, sess)
    cgi_header(
        header_session(sess) +
        header_location("home.web")
    )
} else {
    cgi_header(
        header_location("login.web?error=auth")
    )
}
exit()
}

```

Ketika request method adalah GET, kita cek apakah terdapat query dengan key error. Apabila ya, dan nilainya adalah "auth", kita siapkan error message, yang defaultnya adalah kosong. Lalu, apapun error messagenya, kita tambahkan ke form.

```

var g = cgi_get()
var error = ""
if (g["error"] == "auth") {
    var error = "Authentication failed"
}

```

Yang penting kita bahas adalah ketika request method berupa POST. Kita cek apakah username dan password adalah "admin". Apabila ya, kita anggap otentikasi berhasil. Tentunya, dalam aplikasi nyata, ini mungkin berupa query database.

```

var sess = session_new()
set(sess, "user", u)

```

```

    session_file_set(sess_path, sess)
    cgi_header(
        header_session(sess) +
        header_location("home.web")
    )

```

Dalam hal ini, pembuatan sesi baru dengan fungsi `session_new`, yang akan berupa sebuah HASH. Kita tinggal set key "user". Kemudian, kita simpan file dengan `session_file_set`, dan kirimkan header-header yang diperlukan. Jangan lupa untuk `header_session` (yang akan mengirimkan Set-Cookie).

File `home.web` akan lebih sederhana. Kita tidak perlu bahas secara khusus karena mirip dengan sebelumnya:

```

load_module("web")
var sess_path = "c:\session"
var sess = session_file_get(sess_path)
if ((sess["user"] == null) | (sess["session"] ==
null)) {
    cgi_header(
        header_location("login.web")
    )
    exit()
}

```

```

cgi_header()
print("
    <!DOCTYPE html>
    <html lang='en'>
        <head>
            <title>Home</title>
        </head>
        <body>
            Hello, " + sess["user"] + "

```

```

        <a href='logout.web'>logout</a>
        <hr>
    </body>
</html>
")

```

Lewat link ke `logout.web`, apabila user kunjungi, maka sesi akan dihapus. Berikut adalah isi file `logout.web`:

```

load_module("web")
var sess_path = "c:\session"
var sess = session_file_get(sess_path)
session_file_delete(sess_path, sess)
cgi_header(
    header_session_delete(sess) +
    header_location("login.web")
)

```

Lihatlah bahwa kita perlu hapus file data session dengan `session_file_delete` dan kirimkan header untuk menghapus cookie dengan `header_session_delete`.

Cobalah untuk kunjungi `index.web` dan lakukanlah otentikasi, dan pada akhirnya melakukan `logout`. Kita bisa kunjungi `home.web` berkali-kali dan selama sudah otentikasi dan belum `logout`, kita akan menjumpai halaman `home.web` yang sama. Kini, walau berupa request-request terpisah, terdapat state yang dikelola.

Bab ini sudah selesai? Sebentar lagi. Kita akan akhiri dengan latihan. Cermatilah dengan developer tools web browser yang digunakan, untuk Cookies (misal di bagian Storage).



Ketika belum melakukan otentikasi, tidak terdapat cookie yang diset.

Name	Value	Domain	Path	Expire...	Size
session	70fb3bed771e463b0b0082e46d83b153	localh...	/	Session	39

Ketika otentikasi berhasil dan dibawa ke home.web, terdapat sebuah cookie (dengan nama session) dengan nilai tertentu.

Lihatlah ke direktori di mana file data session disimpan. Sebagai contoh, berikut ini adalah isi file di komputer yang penulis gunakan:



```

{"session": "70fb3bed771e463b0b0082e46d83b153", "user": "admin", "timestamp": 1708610497216}

```

Lakukanlah logout dan bisa kita lihat, cookie tidak lagi tersedia dan file data session pun dihapus.

Catatan tambahan:

- Sebagai alternatif dari fungsi `header_session`, kita juga dapat menggunakan fungsi `header_session_secure` dimana cookie dikirimkan dengan tambahan `Secure`, `HttpOnly`.
- Untuk detail cara kerja module web, Anda mungkin tertarik dengan file `web.singkong` yang disertakan dalam interpreter `Singkong.jar`.

Sampai di sini dulu contoh bekerja dengan session. Di bab berikut, kita akan membahas format data interchange JSON.

Halaman ini sengaja dikosongkan

Mengenal format data-interchange JSON

Dr. Buyung Sofiarto Munir

Teknologi dapat membuat hidup sehari-hari menjadi lebih nyaman. Dengan adanya komputer, pekerjaan bisa dilakukan dengan lebih efisien. Dengan adanya internet, kita dapat belajar, bekerja, dan berkomunikasi tanpa harus terhalang jarak dan waktu. Dan semua ini bahkan menjadi lebih mudah dengan hadirnya ponsel pintar dan aplikasi (software) yang terpasang, yang ikut ke mana pun kita pergi. Tambahkan kecerdasan artifisial di sini, dan kita melihat berbagai peluang baru.

Perkembangan teknologi tersebut memungkinkan lebih banyak software yang dibangun, yang pada akhirnya juga dapat mendorong hadirnya berbagai teknologi baru. Dunia pengembangan software pun tentunya ikut berkembang. Bahasa pemrograman baru, pustaka yang lebih lengkap, alat bantu yang lebih canggih, serta berbagai layanan hadir dan menjadikan hidup programmer ikut lebih nyaman.

Dari sisi bahasa pemrograman saja, setiap dekade, puluhan bahasa baru lahir (walaupun tidak lagi sebanyak misal sekitar 40 tahun lalu). Untuk membuat sebuah software, pilihan bahasanya semakin banyak dan bervariasi. Masing-masing bisa dengan paradigma dan tipe data sendiri, misalnya. Dan tentu saja, sebuah software yang relatif kompleks dapat dibangun oleh sejumlah tim berbeda, dengan bahasa-bahasa yang berbeda pula.

Lalu, bagaimana kalau bagian-bagian dari software tersebut perlu saling bertukar data, ketika secara internal, datanya direpresentasikan secara berbeda? Bagaimana sebuah array di Bahasa Singkong dapat diisi dengan data dari sebuah list di Python?

Bagaimana kalau masing-masing bagian software tersebut juga berjalan di komputer yang berbeda?

Kita dapat mencoba untuk mengharuskan semua bagian dari software tersebut untuk baca dari/tulis ke sistem database yang sama. Tapi, bagaimana kalau ini tidak dimungkinkan secara teknis? Misal tidak tersedia pada sistem operasi komputer yang dimaksud. Lebih lanjut lagi, bagaimana kalau sebuah software perlu bertukar data dengan software lain, tanpa adanya akses ke sistem database yang sama?

Sampai di sini, kita membutuhkan format yang memungkinkan pertukaran data tersebut dilakukan dengan mudah dan dapat dibaca/tulis oleh berbagai bahasa pemrograman. Dengan demikian, selama kebutuhan terpenuhi, kita tidak perlu membuat format sendiri.

Contoh format yang dapat kita gunakan adalah JSON, yang merupakan singkatan dari JavaScript Object Notation (ECMA-404 The JSON Data Interchange Standard). Walaupun berbasiskan pada subset dalam bahasa pemrograman JavaScript, format ini umum didukung oleh berbagai bahasa pemrograman, termasuk bahasa Singkong (walau implementasi dalam Singkong tidak komplit, misal tidak mendukung exponent e pada bilangan, setidaknya pada saat buku ini ditulis).

Data yang ingin dipertukarkan tersebut bisa berupa misal:

- true
- false
- null
- bilangan, seperti 123, -123, 1.23, dan sebagainya.
- string, seperti "halo"
- array, seperti [1,2,3, "empat"]
- object, seperti {"nama": "Singkong", "umur": 5}

Pada berbagai bahasa pemrograman, contoh array sebelumnya mungkin adalah array di sebuah bahasa, dan list pada bahasa lainnya. Sementara, contoh object tersebut adalah hash table atau dictionary di berbagai bahasa. Tipe string dan bilangan tentunya cukup umum, walau bisa dituliskan berbeda antar bahasa. Begitupun juga dengan true dan false, walau literalnya mungkin tidak sama persis. Untuk null, konsep serupa juga umum tersedia, walaupun mungkin dikenal sebagai None, dan sebagainya.

Bisa kita lihat, apabila nilai sebuah variable pada sebuah bahasa bisa dihasilkan dalam format JSON, dan bisa dipahami oleh bahasa lain serta dijadikan sebagai variabel dengan nilai sama, maka pertukaran data telah dilakukan, terlepas dari apapun bahasa pemrograman yang digunakan atau media pertukaran datanya (bisa berupa file pada file sistem, HTTP request, dan lainnya).

Sebagai contoh, katakanlah kita memiliki nilai dict berikut di Python (dikitikkan pada prompt Python):

```
>>> data = {'nama': 'Singkong', 'umur': 5}
>>> data
{'umur': 5, 'nama': 'Singkong'}
```

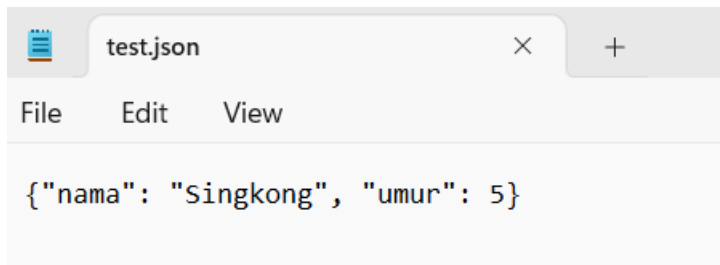
Tentunya, kita tidak dapat mengakses data tersebut di Singkong. Bahkan ketika data tersebut bisa diserialisasi ke sebuah file, atau disimpan sebagai bytecode Python. Kecuali, kalau Singkong dapat memahami format serialisasi atau format bytecode dimaksud (kenyataannya, tidak bisa).

Sekarang, mari kita hasilkan format JSON dari nilai tersebut:

```
>>> import json
>>> open('c:\\data\\test.json',
'w').write(json.dumps(data))
```

Dalam contoh tersebut, kita menggunakan modul bawaan dari Python untuk bekerja dengan format JSON. Kemudian, kita menulis representasi nilai data ke file `c:\data\test.json`. Sesuaikanlah path file ini apabila perlu.

Berikut adalah contoh isi file ketika dibuka dengan notepad:



Bisa kita lihat, mirip sekali. Yang berbeda adalah kutip pada 'nama' yang menjadi "nama", sesuai aturan format.

Sekarang, apabila kita baca file tersebut dari Singkong, kita akan mendapatkan data yang sama sebagai sebuah HASH (dikitikkan pada interactive evaluator Singkong), dengan modul bawaan json:

```
> load_module("json")
> var data = json_parse(read("c:\data\test.json"))
> data
{"nama": "Singkong", "umur": 5}
```

```
 :) Singkong
Interactive Editor Database Help
> load_module("json")
> var data = json_parse(read("c:\data\test.json"))
> data
{"nama": "Singkong", "umur": 5}
```

Bisa kita lihat, nilai data yang tadinya adalah dict di Python bisa didapatkan sebagai HASH di Singkong. Perantaranya dalam hal ini memang sebuah file (test.json), namun tentunya bisa berupa cara lain seperti disebutkan sebelumnya. Kita akan contohkan dalam HTTP request seperti pada bab yang ditulis oleh Dr. Sarwo.

Sebelumnya, kita melihat bahwa terdapat aturan dan nilai yang dapat diterima dalam format JSON. Sebagai contoh, masih di Python (dengan modul json telah diimport), mari kita coba simpan nilai time.struct_time berikut:

```
>>> import time
>>> test = {'time': time.localtime()}
>>> test
{'time': time.struct_time(tm_year=2024, tm_mon=2,
tm_mday=19, tm_hour=13, tm_min=30, tm_sec=27,
tm_wday=0, tm_yday=50, tm_isdst=0)}
>>> json.dumps(test)
```

Pada Python 2.7, kita akan menjumpai TypeError dengan pesan: is not JSON serializable. Pada Python 3, proses dump ini berhasil dan dikonversi ke list.

Bagaimana dengan Singkong? Format JSON tentunya tidak mendukung secara spesifik tipe DATE di Singkong:

```
> type(@)
"DATE"
> var test = {"time": @}
> test
{"time": 2024-02-19 13:40:24}
```

Akan tetapi, fungsi `json_string` dari modul bawaan `json` akan melakukan konversi ke format yang didukung:

```
> load_module("json")
> json_string(test)
"{\"time\": \"2024-02-19 13:40:24\"}"
```

Di Singkong, misal ketika dibaca ulang dari JSON, representasi tanggal dan waktu dalam STRING tersebut bisa dikembalikan ke DATE:

```
> var d = datetime("2024-02-19 13:40:24")
> type(d)
"DATE"
```

Bisa kita lihat, nilai tertentu di luar format yang dikenal oleh JSON perlu ditangani secara tersendiri atau dapat dianggap sebagai error. Ketika ditangani secara tersendiri, tentunya, kita perlu memastikan agar dapat dibaca kembali.

Sampai di sini pembahasan kita. Terima kasih telah membaca.

Form: POST (HASH)

Di catatan akhir bab Form: POST, kita menyebutkan fungsi `cgi_post_hash`. Fungsi ini mirip dengan `cgi_post`, namun alih-alih mengharapkan `key=value` pada body request, `cgi_post_hash` mengharapkan body berupa representasi STRING dari `{key: value}`.

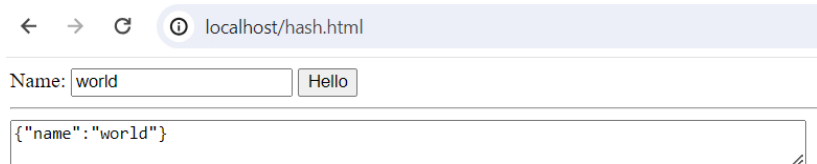
Kita sudah membahas JSON di bab sebelumnya. Apabila body dari request POST adalah representasi STRING dari sebuah HASH, atau sampai batasan tertentu, adalah sebuah data JSON, fungsi `cgi_post` tidaklah tepat digunakan (karena mengharapkan pola `key=value`). Di sinilah `cgi_post_hash` akan membantu.

Mari kita siapkan contoh `hash.html` berikut (isinya akan kita sesuaikan lagi nanti):

```
<!DOCTYPE html>
<html lang='en'>
  <head>
    <title>Hash</title>
  </head>
  <body>
    Name: <input id='name'>
    <button onclick='req()'>Hello</button>
    <hr>
    <textarea id='result' cols='80'></textarea>
    <script>
      function req() {
        var n =
document.getElementById('name').value;
        var d = JSON.stringify({"name": n});
        document.getElementById('result').value =
d;
      }
    </script>
  </body>
```

</html>

Dalam hal ini, setelah name diisi dan tombol Hello di klik, kita menampilkan data JSON dari name tersebut:



The screenshot shows a web browser window with the address bar displaying 'localhost/hash.html'. Below the address bar, there is a form with a text input field containing 'world' and a 'Hello' button. Below the form, there is a text area containing the JSON response: {"name": "world"}.

Kita tidak menggunakan submit biasa karena perlu mengkonversi ke JSON terlebih dahulu. Saat ini, apabila nilai {"name": "world"} adalah body dari request POST, kita perlu tangani dengan `cgi_post_hash`.

Mari kita sesuaikan `hash.html` sebelumnya dengan menambahkan kode untuk melakukan request, mendapatkan hasil, dan menampilkan dalam `textarea`:

```
<!DOCTYPE html>
<html lang='en'>
  <head>
    <title>Hash</title>
  </head>
  <body>
    Name: <input id='name'>
    <button onclick='req()'>Hello</button>
    <hr>
    <textarea id='result' cols='80'></textarea>
    <script>
      function req() {
        var n =
document.getElementById('name').value;
        var d = JSON.stringify({"name": n});
        var xhr = new XMLHttpRequest();
        xhr.open("POST",
"http://localhost/hash.web", true);
```

```

        xhr.onreadystatechange = function() {
            if (xhr.readyState === 4 && xhr.status
=== 200) {
                document.getElementById('result').value
= xhr.responseText;
            }
        }
        xhr.send(d)
    }
</script>
</body>
</html>

```

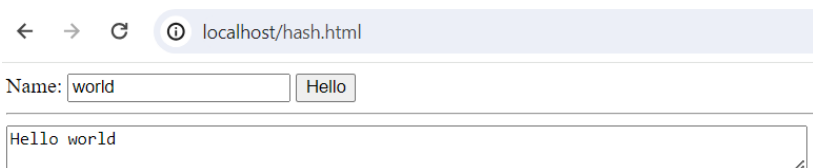
Kita siapkan hash.web dengan isi file berikut (perhatikanlah fungsi `cgi_post_hash` yang digunakan):

```

cgi_header()
var g = cgi_post_hash()
var n = g["name"]
var m = "Hello "
if (n != null) {
    var m = m + n
}
println(m)

```

Kunjungi <http://localhost/hash.html>, isikanlah name, dan kliklah tombol Hello.



The screenshot shows a web browser window with the address bar displaying 'localhost/hash.html'. Below the address bar, there is a form with a 'Name:' label, a text input field containing the text 'world', and a 'Hello' button. Below the form, there is a text area containing the text 'Hello world'.

Mirip dengan contoh sebelumnya, hanya saja kita gunakan XMLHttpRequest dengan body berupa JSON. Dalam hal ini, hash.web

mencetak STRING. Andaikata kita ubah hash.web menjadi berikut agar mencetak JSON:

```
load_module("json")
cgi_header({"Content-type": "application/json"})
var g = cgi_post_hash()
var n = g["name"]
var m = "Hello "
if (n != null) {
    var m = m + n
}
print(json_string({"message": m}))
```

Dan kita sesuaikan hash.html menjadi (penyesuaian diformat tebal):

```
<!DOCTYPE html>
<html lang='en'>
  <head>
    <title>Hash</title>
  </head>
  <body>
    Name: <input id='name'>
    <button onclick='req()'>Hello</button>
    <hr>
    <textarea id='result' cols='80'></textarea>
    <script>
      function req() {
        var n =
document.getElementById('name').value;
        var d = JSON.stringify({"name": n});
        var xhr = new XMLHttpRequest();
        xhr.open("POST",
"http://localhost/hash.web", true);
        xhr.onreadystatechange = function() {
          if (xhr.readyState === 4 && xhr.status
=== 200) {
            var r = JSON.parse(xhr.responseText);

```

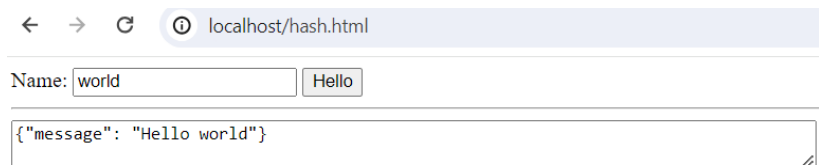
```

        document.getElementById('result').value
= r.message;
    }
}
xhr.send(d)
}
</script>
</body>
</html>

```

Kembalilah kunjungi <http://localhost/hash.html>, isikanlah name, dan kliklah tombol Hello. Hasilnya akan terlihat sama seperti sebelumnya, tapi alih-alih sekedar menampilkan teks kembalian hash.web, kita melakukan parsing JSON dengan `JSON.parse` JavaScript, kemudian menampilkan message.

Andaikata hash.html masih sama seperti sebelumnya (`document.getElementById('result').value = xhr.responseText`), maka contoh tampilannya akan berbeda:



Memang agak lebih repot, Anda mungkin berpendapat. Tentunya penulis juga sependapat. Tapi, mari kita lakukan request POST dengan body JSON tanpa browser, ke hash.web, seperti contoh berikut (dapat diketikkan pada Interactive evaluator Singkong):

```

> load_module("json")
> var r = http_post("http://localhost/hash.web",
json_string({"name": "world"}))
> var m = json_parse(r[2])
> m

```

```
{"message": "Hello world"}
```

```
> m["message"]  
"Hello world"
```



```
;) Singkong  
Interactive Editor Database Help  
  
> load_module("json")  
> var r = http_post("http://localhost/hash.web", json_string({"name": "world"}))  
> var m = json_parse(r[2])  
> m  
{"message": "Hello world"}  
  
> m["message"]  
"Hello world"
```

Dengan `cgi_post_hash`, kita otomatis mendapatkan HASH dari body berupa representasi STRING HASH (misal JSON). Sampai di sini dulu pembahasan kita.

Mari kita lihat contoh lain dari JSON di bab berikut.

Contoh HTTP Request dengan JSON

Dr. Sarwo

Pada pengenalan format data-interchange JSON yang ditulis oleh Dr. Buyung, kita melihat contoh bagaimana dict pada Python disimpan dalam format JSON ke sebuah file, yang kemudian dibaca dari Singkong dan dijadikan sebagai sebuah HASH. Bahasa-bahasa berbeda, namun dapat saling bertukar data.

Dalam berbagai aplikasi, terutama yang melakukan pemanggilan API lewat HTTP, kita tidak berbagi file yang sama. Hal ini diantaranya karena masing-masing dijalankan dari komputer yang berbeda. Karena kita sudah berkomunikasi lewat HTTP, maka pertukaran data dilakukan sebagai sebuah HTTP request dengan data berupa format JSON, yang diterima dan diproses oleh server, dan dikembalikan sebagai HTTP response, juga dalam format JSON. Dalam hal ini, baik server dan client tentunya juga dapat ditulis dengan bahasa-bahasa pemrograman yang berbeda.

Di dalam pembahasan ini, kita akan menyajikan contoh di mana di sisi HTTP server, kita memiliki sebuah program yang ditulis dengan bahasa Singkong, yang:

- Menerima HTTP request dengan method POST, sekedar untuk ilustrasi body berupa data JSON. Tentu saja, untuk kebutuhan pembahasan ini, kita juga dapat menggunakan method GET (yang tampaknya lebih sesuai, apabila melihat point berikut).
- Dengan isi berupa data dalam format JSON, berupa sebuah object dengan key adalah "number" dan value berupa bilangan dalam string. Misal: "number": "12345.678".
- Program di sisi server akan mengembalikan HTTP response dengan Content-type: application/json, juga berupa sebuah object, dengan key adalah "words" dan value berupa nilai

terbilang dalam Bahasa Indonesia. Misal: "words": "Dua belas ribu tiga ratus empat puluh lima koma enam tujuh delapan".

Pada awal buku, kita sudah melihat contoh bagaimana request dengan method POST diproses. Kita akan menggunakan contoh serupa, dengan kode program sebagai berikut, dan disimpan sebagai file `words.web`, yang dapat diakses lewat URL: `http://localhost/words.web`

```
load_module("json")

cgi_header({"Content-Type": "application/json"})
var p = cgi_post_hash()
var n = p["number"]
var r = {"words": ""}
if (n != null) {
    set(r, "words", words_id(n))
}
println(r)
```

Sekarang, mari kita siapkan di sisi client. Tanpa menulis program misalnya, kita bisa menggunakan program untuk melakukan HTTP request dengan method POST, seperti `curl`. Apabila terinstal, kita bisa mencoba melakukan request dengan contoh berikut:

```
$ curl -s -X POST -d '{"number": "12345.678"}'
http://localhost/words.web

{"words": "Dua belas ribu tiga ratus empat puluh
lima koma enam tujuh delapan"}
```


Tentu saja, dalam hal ini, kita mendapatkan HTTP response yang benar, namun tidak kita proses secara langsung. Kita tahu bahwa ini adalah sebuah object dalam JSON, dengan key adalah "words" dan value yang sesuai.

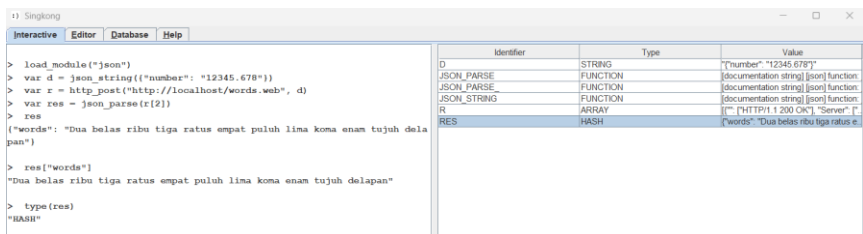
Mari kita gunakan Singkong untuk melakukan request dan mendapatkan hasil (diketikkan pada Interactive evaluator):

```
> load_module("json")
> var d = json_string({"number": "12345.678"})
> var r = http_post("http://localhost/words.web",
d)
> var res = json_parse(r[2])
> res
{"words": "Dua belas ribu tiga ratus empat puluh
lima koma enam tujuh delapan"}

> res["words"]
"Dua belas ribu tiga ratus empat puluh lima koma
enam tujuh delapan"

> type(res)
"HASH"
```

Bisa kita lihat, hasil yang sama kita dapatkan, namun telah berupa sebuah HASH, sebagaimana kita harapkan.



Sekarang, bagaimana kalau program pemanggil tidak ditulis dengan Singkong? Berikut adalah contoh dengan Python 2.7, hanya dengan standard library.

```
>>> import json
>>> import urllib2
>>> d = json.dumps({'number': '12345.678'})
>>> r =
urllib2.urlopen('http://localhost/words.web',
d).read()
>>> res = json.loads(r)
>>> res
{u'words': u'Dua belas ribu tiga ratus empat puluh
lima koma enam tujuh delapan'}
>>> res['words']
u'Dua belas ribu tiga ratus empat puluh lima koma
enam tujuh delapan'
>>>
```

Dengan Python 3, juga hanya dengan standard library:

```
>>> import json
>>> import urllib.request
>>> d = json.dumps({'number':
'12345.678'}).encode('utf-8')
>>> req =
urllib.request.Request('http://localhost/words.web'
, d, {})
>>> r =
urllib.request.urlopen(req).read().decode('utf-8')
>>> res = json.loads(r)
>>> res
{'words': 'Dua belas ribu tiga ratus empat puluh
lima koma enam tujuh delapan'}
>>> res['words']
'Dua belas ribu tiga ratus empat puluh lima koma
enam tujuh delapan'
>>>
```

Apabila dengan GET, kita cukup menggunakan query string pada URL menggunakan web browser. Namun, dengan POST dan body berupa JSON, mari kita contohkan dengan HTML dan JavaScript, yang disimpan pada words.html dan dapat diakses dengan web browser lewat `http://localhost/words.html` (diadaptasi cari contoh di bab sebelumnya):

Berikut adalah isi file words.html:

```
<!DOCTYPE html>
<html lang='en'>
  <head>
    <title>Words</title>
  </head>
  <body>
    Number: <input id='number' value='12345.678'>
    <button onclick='req()'>Request</button>
    <hr>
    <textarea id='result' cols='80'></textarea>
    <script>
      function req() {
        var n = document.getElementById('number').value;
        var d = JSON.stringify({"number": n});
        var xhr = new XMLHttpRequest();
        xhr.open("POST",
"http://localhost/words.web", true);
        xhr.onreadystatechange = function() {
          if (xhr.readyState === 4 && xhr.status ===
200) {
            var r = JSON.parse(xhr.responseText);
            document.getElementById('result').value
= r.words;
          }
        }
        xhr.send(d)
      }
    </script>
```

```
</body>  
</html>
```

A screenshot of a web browser interface. The address bar shows 'localhost/words.html'. Below the address bar, there is a 'Number:' label followed by an input field containing '12345.678' and a 'Request' button. Below this, a text area displays the response: 'Dua belas ribu tiga ratus empat puluh lima koma enam tujuh delapan'.

Dari contoh-contoh tersebut, bisa kita lihat bahwa kita telah melakukan pertukaran data dalam format JSON, di mana bahasa yang digunakan untuk menulis program di sisi server dan client dapat berbeda.

HTTP Client

Dengan program di sisi HTTP server menghasilkan format pertukaran data tertentu, kita dapat membangun client yang berjalan di berbagai platform/device, tanpa menggunakan web browser.

Singkong datang dengan sejumlah fungsi bawaan untuk melakukan request lewat HTTP, seperti dirinci dalam tabel berikut:

Method	Fungsi	Catatan
HEAD	http_head	Redireksi HTTP ke HTTPS didukung. Hanya sekali redirect (Location) yang akan di-follow. Status code redirect berikut didukung: 301, 302, 303.
GET	http_get http_get_file	Redirect sama seperti pada http_head. Fungsi http_get_file dapat digunakan untuk mendownload file.
POST	http_post http_post_override	Content-Type: application/x-www-form-urlencoded;charset=UTF-8 Body kosong diijinkan. Untuk POST dengan X-HTTP-Method-Override, gunakanlah fungsi http_post_override
PUT	http_put	Untuk content-type dan body sama seperti pada POST
DELETE	http_delete	Content-Type: application/x-www-form-urlencoded;charset=UTF-8 Body tidak disupport

Untuk method yang tidak disupport secara langsung, apabila HTTP server mendukung X-HTTP-Method-Override, fungsi bawaan `http_post_override` mungkin dapat digunakan (misal untuk method PATCH).

Fungsi-fungsi dalam tabel mengembalikan NULL apabila terjadi error, dan ARRAY (headers, response code, data) apabila berhasil. Selain itu, untuk charset, fungsi-fungsi tersebut mengirimkan `Accept-Charset: UTF-8`.

Mari kita lihat kembali contoh `hash.web` sebelumnya. Kita akan bekerja dengan JSON, sehingga load module bawaan `json` diperlukan:

```
> load_module("json")
```

Dalam contoh tersebut, kita melakukan request dengan method POST, ke `http://localhost/hash.web`. Fungsi yang digunakan adalah `http_post`. Fungsi ini menerima argumen berupa STRING (URL) and STRING (body). Oleh karena itu, kita memanggil `http_post` tersebut dengan:

```
> var r = http_post("http://localhost/hash.web",  
json_string({"name": "world"}))
```

Apabila gagal, tentunya `r` akan bernilai NULL. Dalam hal ini, ketika berhasil, berikut adalah contoh di komputer yang penulis gunakan:

```
> r  
[{"": ["HTTP/1.1 200 OK"], "Server": ["Microsoft-IIS/10.0"], "Connection": ["close"], "Content-Length": ["26"], "Date": ["Fri, 23 Feb 2024 06:11:18 GMT"], "Content-Type": ["application/json"]}, 200,
```

```
"{"message": "Hello world"}"]
```

Bisa kita lihat, ini adalah sebuah ARRAY (headers, response code, data):

```
> r[0]
{"": ["HTTP/1.1 200 OK"], "Server": ["Microsoft-IIS/10.0"], "Connection": ["close"], "Content-Length": ["26"], "Date": ["Fri, 23 Feb 2024 06:11:18 GMT"], "Content-Type": ["application/json"]}
```

```
> r[1]
200
```

```
> r[2]
{"message": "Hello world"}"
```

Kita menggunakan fungsi `json_parse` untuk parsing JSON:

```
> var m = json_parse(r[2])
> m
{"message": "Hello world"}
```

Kini, `m` adalah sebuah HASH:

```
> type(m)
"HASH"
> m["message"]
"Hello world"
```

Untuk request yang sukses (mengembalikan ARRAY (headers, response code, data)), kita dapat menggunakan fungsi

`http_response_ok` yang, apabila status code response adalah 200, akan mengembalikan data. Sebaliknya, mengembalikan NULL. Contoh penerapan pada contoh sebelumnya:

```
> load_module("json")
> var r = http_post("http://localhost/hash.web",
  json_string({"name": "world"}))
> var d = http_response_ok(r)
> d
"{\"message\": \"Hello world\"}"
> json_parse(d)
{"message": "Hello world"}
```

Untuk fungsi-fungsi lain dalam tabel, cara penggunaannya akan mirip. Apabila sempat, cobalah terapkan untuk contoh-contoh yang kita bahas sebelumnya.

Sebelum mengakhiri buku ini, kita akan membahas beberapa hal berikut, yang mungkin berguna.

Contoh parsing JSON

Apabila akan sering bekerja dengan data JSON sebagai response dari sisi server, beberapa contoh berikut mungkin akan berguna.

Untuk bekerja dengan JSON di Singkong, kita umumnya menggunakan fungsi `json_string` untuk menghasilkan STRING JSON, dan `json_parse` untuk parsing JSON ke tipe Singkong. Fungsi-fungsi tersebut disertakan dalam modul `json`, sehingga kita perlu `load_module("json")` terlebih dahulu.

Untuk parsing, apabila terjadi kegagalan, maka STRING kosong akan dikembalikan. Berikut adalah beberapa contohnya. Kita akan mulai dengan null, true, false, dan bilangan:

```
> type(json_parse("null"))
"NULL"
> json_parse("true")
true
> json_parse("false")
false
> json_parse("1.23")
1.2300
```

Untuk ARRAY:

```
> json_parse("[null, true, 1.23, {}]")
[null, true, 1.2300, {}]
```

Contoh gagal karena key harus berupa STRING:

```
> json_parse("{1: 2, true: []}")
""
```

Disarankan untuk menggunakan `json_string`, bukan fungsi bawaan string, untuk penambahan kutip misalnya:

```
> json_parse(string({1: 2, true: []}))
""
> json_parse(json_string({1: 2, true: []}))
{"1": 2, "true": []}
> json_string({1: [null, true, {}], 2: 3})
```

```
"{"1": [null, true, {}], "2": 3}"
```

Bisa kita lihat, data berupa test berikut (4 karakter, tanpa kutip) tidaklah valid. Diperlukan penambahan kutip.

```
> json_parse("test")
""
> json_parse(json_string("test"))
"test"
```

Catatan: Sebagaimana telah disebutkan sebelumnya, implementasi JSON dalam Singkong tidak komplit, misal tidak mendukung exponent e pada bilangan, setidaknya pada saat buku ini ditulis.

Base64

Apabila perlu bekerja dengan Base64, Singkong menyediakan fungsi-fungsi bawaan berikut: `base64_decode`, `base64_encode`, `x_base64_decode_file`, dan `x_base64_encode_file`. Fungsi yang namanya diawali oleh `x_` menandakan fungsi extended dan membutuhkan Java versi > 5.0.

Encode/Decode

Untuk melakukan encode/decode secara manual terhadap STRING application/x-www-form-urlencoded, gunakanlah fungsi-fungsi bawaan `url_encode` dan `url_decode`.

Timeout dan header

Fungsi-fungsi dalam tabel sebelumnya menerima argumen opsional, dimana timeout diberikan dalam milidetik dan header dilewatkan sebagai HASH.

Contoh HTTP client dan GUI

Sebagai contoh, kita akan melakukan request POST ke server, dengan tujuan simulasi otentikasi (dengan username dan password), dengan body `username=<string>&password=<string>` dan server akan memberikan response dalam format JSON. Untuk sisi GUI, kita bisa menggunakan fungsi `login_dialog`.

Kita siapkan di sisi server, dengan file `auth.web` berikut:

```
load_module("json")

var h = {"Content-Type": "application/json"}
var r = {"result": false}
var p = cgi_post()
var x = p["username"]
var y = p["password"]

if (is(x, "STRING") & is(y, "STRING")) {
    set(r, "result", x == "admin" & y == "admin")
}

cgi_header(h)
println(json_string(r))
```

Apabila kita kunjungi dengan browser (GET), tentu saja kita akan mendapatkan result false:

```
{"result": false}
```

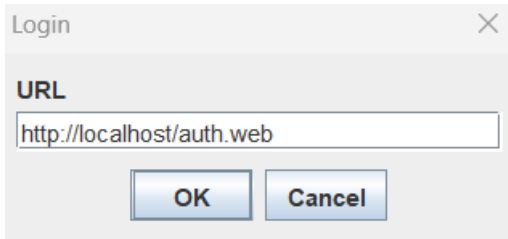
Mari kita siapkan client, yang disimpan sebagai file `login.singkong` (program GUI biasa, dengan ekstensi nama file `.singkong`):

```
load_module("json")

var url = trim(input("URL", "Login"))
if (!startswith(url, "http")) {
    exit()
}

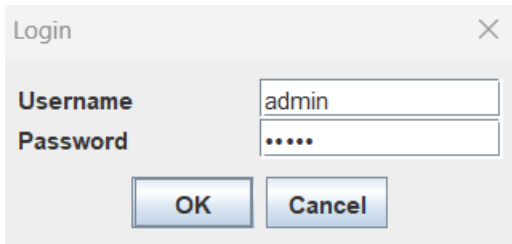
var login = login_dialog("Login")
if (len(login) == 2) {
    var u = login[0]
    var p = login[1]
    var data = "username=" + u + "&password=" + p
    var res = http_post(url, data)
    if (res != null) {
        if (is(res, "ARRAY")) {
            if (len(res) == 3) {
                var r = json_parse(res[2])
                message(r["result"])
            }
        }
    }
}
```

Ketika `login.singkong` dijalankan, masukkanlah alamat `http://localhost/auth.web`:

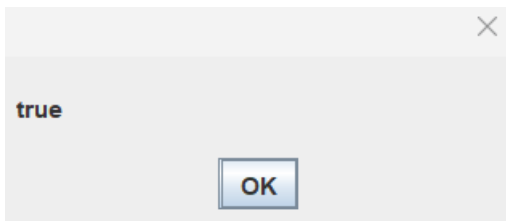


Kemudian, kliklah tombol OK.

Form login akan ditampilkan, masukkanlah admin untuk username dan password, kemudian kliklah tombol OK:



Sebuah message box akan ditampilkan dengan isi sesuai response yang didapatkan dari auth.web:



Halaman ini sengaja dikosongkan

Daftar Pustaka

Barth, A., "HTTP State Management Mechanism", RFC 6265, DOI 10.17487/RFC6265, April 2011, <<https://www.rfc-editor.org/info/rfc6265>>.

Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<https://www.rfc-editor.org/info/rfc3986>>.

Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, DOI 10.17487/RFC2616, June 1999, <<https://www.rfc-editor.org/info/rfc2616>>.

Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <<https://www.rfc-editor.org/info/rfc9110>>.

Noprianto., 2024, Mengenal dan Menggunakan Bahasa Pemrograman Singkong, PT. Stabil Standar Sinergi.

Noprianto, Iskandar, K., and Soewito, B., 2022, Contoh dan Penjelasan Bahasa Singkong: Dasar-dasar Aplikasi GUI, PT. Stabil Standar Sinergi.

Robinson, D. and K. Coar, "The Common Gateway Interface (CGI) Version 1.1", RFC 3875, DOI 10.17487/RFC3875, October 2004, <<https://www.rfc-editor.org/info/rfc3875>>.

Buku ini berisi sejumlah contoh source code dan penjelasan langkah demi langkah yang mudah dipahami untuk membuat aplikasi web, dengan bahasa pemrograman Singkong.

Contoh-contoh yang dibahas mencakup instalasi dan konfigurasi HTTP server, pengenalan HTTP dan CGI, menangani request GET dan POST, bekerja dengan session, JavaScript Object Notation (JSON), dan HTTP client.



Dr. Noprianto mengembangkan bahasa pemrograman Singkong dan interpreturnya sejak akhir 2019.

Beliau menyukai pemrograman, dan mendirikan serta mengelola perusahaan pengembangan software dan teknologi Singkong.dev (PT. Stabil Standar Sinergi).

Noprianto menyelesaikan pendidikan doktor ilmu komputer dan telah menulis beberapa buku pemrograman (termasuk Python, Java, dan Singkong).

Buku dan softwarena dapat didownload dari <https://nopri.github.io>



Dr. Buyung Sofiarto Munir saat ini adalah Vice President Jaringan dan Sistem di PT. PLN Enjiniring, setelah sebelumnya mengemban berbagai tanggung jawab manajerial di PT. PLN (Persero) selama 20 tahun.

Selain sebagai seorang praktisi berpengalaman, beliau juga adalah dosen dan peneliti dengan 45 publikasi ilmiah di berbagai konferensi dan jurnal internasional.

Dr. Buyung memiliki gelar Doktor di bidang ilmu komputer dari Universitas Bina Nusantara, Master of Science dari Delft University of Technology, dan Sarjana Teknik Elektro dari Institut Teknologi Bandung.



Dr. Sarwo adalah seorang praktisi, dosen, dan peneliti di bidang teknologi informasi.

Saat ini, selain sebagai dosen di Universitas Bina Nusantara, beliau juga menjadi konsultan pengembangan software di beberapa lembaga.

Dr. Sarwo memiliki gelar Doktor di bidang ilmu komputer dari Universitas Bina Nusantara, dan telah memiliki publikasi ilmiah di beberapa konferensi dan jurnal internasional.

Fokus penelitian beliau saat ini adalah pada deep learning dan computer vision.

singkong.dev

Alamat: Puri Indah Financial Tower
Lantai 6, Unit 0612
Jl. Puri Lingkar Dalam Blok T8
Kembangan, Jakarta Barat 11610
Email: info@singkong.dev

ISBN 978-602-52770-6-1

