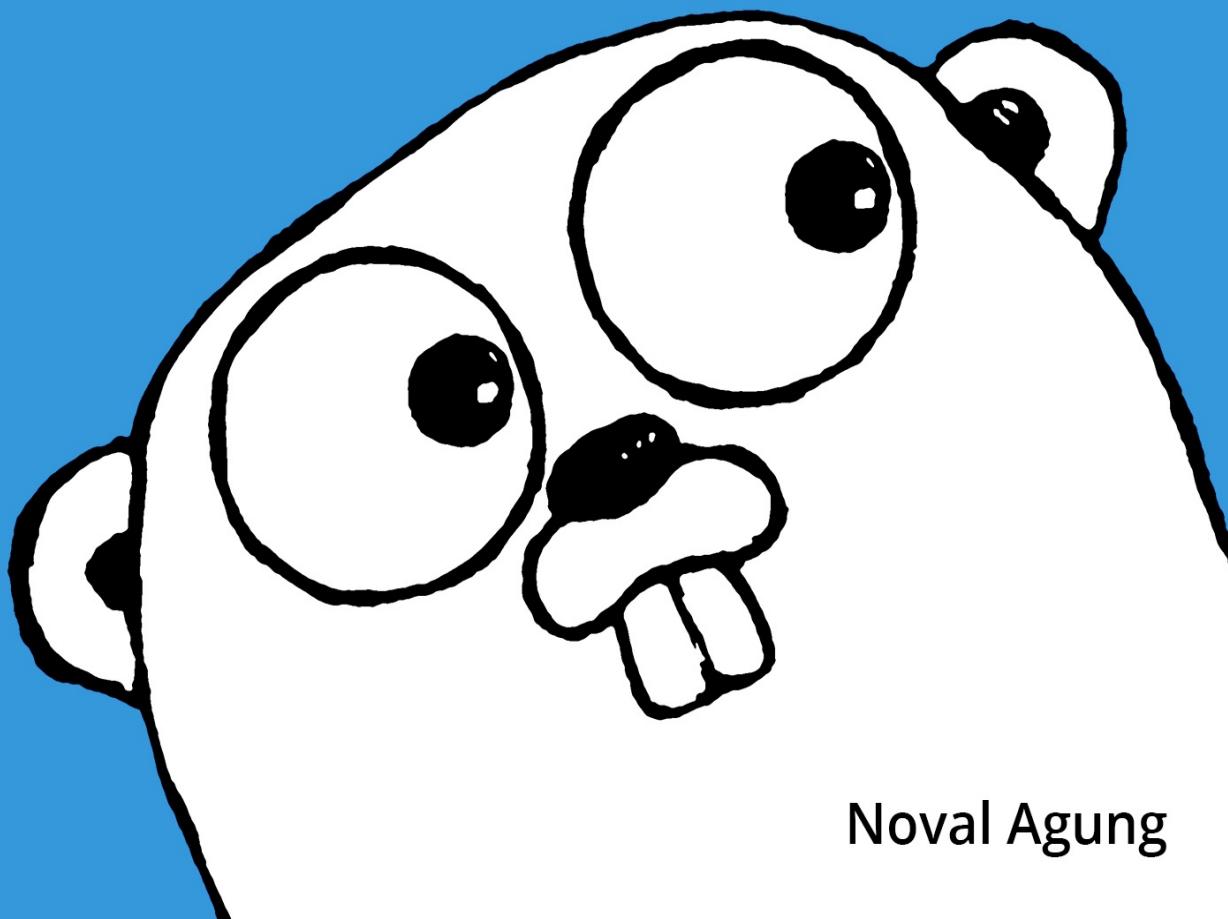


# DASAR PEMROGRAMAN

# GOLANG



Noval Agung

# Table of Contents

Introduction	1.1
A. Pemrograman Golang Dasar	1.2
A.1. Berkenalan Dengan Golang	1.2.1
A.2. Instalasi Golang	1.2.2
A.3. GOPATH Dan Workspace	1.2.3
A.4. Instalasi Editor	1.2.4
A.5. Command	1.2.5
A.6. Program Pertama: Hello World	1.2.6
A.7. Komentar	1.2.7
A.8. Variabel	1.2.8
A.9. Tipe Data	1.2.9
A.10. Konstanta	1.2.10
A.11. Operator	1.2.11
A.12. Seleksi Kondisi	1.2.12
A.13. Perulangan	1.2.13
A.14. Array	1.2.14
A.15. Slice	1.2.15
A.16. Map	1.2.16
A.17. Fungsi	1.2.17
A.18. Fungsi Multiple Return	1.2.18
A.19. Fungsi Variadic	1.2.19
A.20. Fungsi Closure	1.2.20
A.21. Fungsi Sebagai parameter	1.2.21
A.22. Pointer	1.2.22
A.23. Struct	1.2.23
A.24. Method	1.2.24
A.25. Property Public & Private	1.2.25
A.26. Interface	1.2.26
A.27. Interface Kosong	1.2.27
A.28. Reflect	1.2.28
A.29. Goroutine	1.2.29
A.30. Channel	1.2.30
A.31. Buffered Channel	1.2.31
A.32. Channel - Select	1.2.32
A.33. Channel - Range & Close	1.2.33
A.34. Channel - Timeout	1.2.34
A.35. Defer & Exit	1.2.35

---

A.36. Error, Panic, & Recover	1.2.36
A.37. Layout Format String	1.2.37
A.38. Time, Parsing Time, & Format Time	1.2.38
A.39. Timer	1.2.39
A.40. Konversi Antar Tipe Data	1.2.40
A.41. Fungsi String	1.2.41
A.42. Regex	1.2.42
A.43. Encode - Decode Base64	1.2.43
A.44. Hash Sha1	1.2.44
A.45. Arguments & Flag	1.2.45
A.46. Exec	1.2.46
A.47. File	1.2.47
A.48. Web	1.2.48
A.49. URL Parsing	1.2.49
A.50. JSON	1.2.50
A.51. Web JSON API	1.2.51
A.52. HTTP Request	1.2.52
A.53. SQL	1.2.53
A.54. NoSQL MongoDB	1.2.54
A.55. Unit Test	1.2.55
A.56. WaitGroup	1.2.56
A.57. Mutex	1.2.57
A.58. Go Vendoring	1.2.58
A.59. Dep - Go Dependency Management Tool	1.2.59
A.60. Go Modules	1.2.60
B. Pemrograman Web Golang Dasar	1.3
B.1. Golang Web App: Hello World	1.3.1
B.2. Routing http.HandleFunc	1.3.2
B.3. Routing Static Assets	1.3.3
B.4. Template: Render HTML Template	1.3.4
B.5. Template: Render Partial HTML Template	1.3.5
B.6. Template: Actions & Variables	1.3.6
B.7. Template: Functions	1.3.7
B.8. Template: Custom Functions	1.3.8
B.9. Template: Render Specific HTML Template	1.3.9
B.10. Template: Render HTML String	1.3.10
B.11. HTTP Method: POST & GET	1.3.11
B.12. Form Value	1.3.12
B.13. Form Upload File	1.3.13
B.14. AJAX JSON Payload	1.3.14

---

---

B.15. AJAX JSON Response	1.3.15
B.16. AJAX Multiple File Upload	1.3.16
B.17. Download File	1.3.17
B.18. HTTP Basic Auth	1.3.18
B.19. Middleware http.Handler	1.3.19
B.20. Custom Multiplexer	1.3.20
B.21. HTTP Cookie	1.3.21
B.22. Configuration File	1.3.22
<b>C. Pemrograman Web Golang Lanjut</b>	<b>1.4</b>
C.1. Echo Framework & Routing	1.4.1
C.2. Parsing HTTP Request Payload (Echo)	1.4.2
C.3. HTTP Request Payload Validation (Validator v9, Echo)	1.4.3
C.4. HTTP Error Handling (Validator v9, Echo)	1.4.4
C.5. Template Rendering in Echo	1.4.5
C.6. Advanced Middleware & Logging (Logrus, Echo Logger)	1.4.6
C.7. CLI Flag Parser (Kingpin)	1.4.7
C.8. Advanced Configuration File (Viper)	1.4.8
C.9. Secure Cookie (Gorilla Securecookie)	1.4.9
C.10. Session (Gorilla Session)	1.4.10
C.12. CORS & Preflight Request	1.4.11
C.13. CSRF	1.4.12
C.14. Secure Middleware	1.4.13
C.15. HTTP Gzip Compression (gziphandler)	1.4.14
C.16. Send Mail (net/smtp, Gomail v2)	1.4.15
C.17. Read & Write Excel XLSX File (Excelize)	1.4.16
C.18. Write PDF File (gofpdf)	1.4.17
C.19. Convert HTML to PDF (go-wkhtmltopdf)	1.4.18
C.20. Scraping & Parsing HTML (goquery)	1.4.19
C.21. Parse & Generate XML (etree)	1.4.20
C.22. HTTPS/TLS Web Server	1.4.21
C.23. HTTP/2 & HTTP/2 Server Push	1.4.22
C.24. Client HTTP Request	1.4.23
C.25. Secure & Insecure Client HTTP Request	1.4.24
C.26. FTP	1.4.25
C.27. SSH & SFTP	1.4.26
C.28. Web Socket: Chatting App	1.4.27
C.29. Protobuf	1.4.28
C.30. gRPC + Protobuf	1.4.29
C.31. Context: Value, Timeout, & Cancellation	1.4.30
C.32. JSON Web Token (JWT)	1.4.31

---



# Dasar Pemrograman Golang

Golang (atau Go) adalah bahasa pemrograman yang lahir di tahun 2009. Golang memiliki banyak kelebihan, terbukti dengan banyaknya perusahaan besar yang menggunakan bahasa ini dalam pengembangan produk-produk mereka, hingga level production tentunya.

Ebook ini merupakan salah satu dari sekian banyak referensi yang bisa dijadikan bahan belajar pemrograman Golang. Topik-topik yang disediakan sangat bervariasi mulai dari hal yang basic (dari 0), hingga bab yang sifatnya advance.

Ada total sekitar **110 bab** yang dibahas dalam ebook ini. Bab-bab tersebut dibagi menjadi 3 kategori besar yang berurutan dan berkesinambungan satu sama lain.

- A. **Pemrograman Golang Dasar.** Pada bagian ini topik yang dibahas sangat dasar, cocok untuk orang yang belum pernah tau atau belum menggunakan bahasa golang. Pembahasan dimulai dari instalasi, eksekusi, hello word, dilanjutkan dengan topik seperti pembahasan beberapa keyword golang, pointer, struct, interface, reflect, goroutine, channel, date time, dan lainnya.
- B. **Pemrograman Web Golang Dasar.** Pada bagian ini kita akan fokus belajar ilmu dasar yang diperlukan untuk pengembangan aplikasi web menggunakan golang, diantaranya seperti: routing, multiplexer, middleware, cookie, dan lainnya. Pada bab ini kita tidak menggunakan framework atau library external, hanya menggunakan API internal yang disediakan golang saja.
- C. **Pemrograman Web Golang Lanjut.** Di bagian ini akan mulai dibahas topik yang lebih advance, beberapa diantaranya akan menggunakan library-library golang yang sudah cukup terkenal di komunitas. Topik-topik tersebut antara lain: http, ssl, cors, csrf, mail, pdf, excel, ftp, ssh, web socket, protobuf, hingga gRPC + protobuf.

Ebook Dasar Pemrograman Golang gratis untuk disebarluaskan secara bebas, selama tidak melanggar aturan lisensi [GNU LGPL 2.1](#).

Source code contoh-contoh program bisa diunduh di [github.com/novalagung](#). Dianjurkan untuk tidak copy-paste dari source code dalam belajar, usahakan untuk menulis sendiri kode program, agar cepat terbiasa dengan bahasa Golang.

## Versi Ebook: 2.2019.03.26

Konten pada ebook ini telah di update menyesuaikan go versi 1.11.

Ebook ini bisa di-download dalam bentuk [PDF](#), [ePub](#), dan [mobi](#). Untuk mendapatkan konten buku yang paling update, silakan baca online atau download ulang ebook. Konten pada versi terbaru lebih update.

Ebook ini dibuat oleh **Noval Agung Prayogo**. Untuk pertanyaan, kritik, dan saran, silakan drop email ke [caknopal@gmail.com](mailto:caknopal@gmail.com).

## A.1. Berkenalan Dengan Golang

**Golang** (atau biasa disebut dengan **Go**) adalah bahasa pemrograman baru yang dikembangkan di **Google** oleh **Robert Griesemer, Rob Pike**, dan **Ken Thompson** pada tahun 2007 dan mulai diperkenalkan di publik tahun 2009.

Penciptaan bahasa Golang didasari bahasa **C** dan **C++**, oleh karena itu gaya sintaks-nya mirip.

### A.1.1. Kelebihan Golang

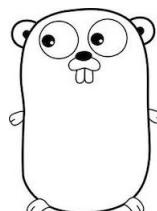
Golang memiliki kelebihan dibanding bahasa lainnya, beberapa di antaranya:

- Mendukung konkurenси di level bahasa dengan pengaplikasian cukup mudah
- Mendukung pemrosesan data dengan banyak prosesor dalam waktu yang bersamaan (*parallel processing*)
- Memiliki garbage collector
- Proses kompilasi sangat cepat
- Bukan bahasa pemrograman yang hirarkial, menjadikan developer tidak perlu *ribet* memikirkan segmen OOP-nya
- Package/modul yang disediakan terbilang lengkap. Karena bahasa ini open source, banyak sekali developer yang juga mengembangkan modul-modul lain yang bisa dimanfaatkan

Sudah banyak industri dan perusahaan yg menggunakan Golang sampai level production, termasuk diantaranya adalah Google sendiri, dan tempat dimana saya bekerja

---

Di buku ini kita akan belajar tentang dasar pemrograman Golang mulai dari 0.



## A.2. Instalasi Golang

Hal pertama yang perlu dilakukan sebelum bisa menggunakan Golang adalah meng-install-nya terlebih dahulu. Panduan instalasi sebenarnya sudah disediakan di situs official Golang <http://golang.org/doc/install#install>.

Disini penulis mencoba meringkas petunjuk instalasi di link tersebut, agar lebih mudah untuk diikuti terutama untuk pembaca yang baru belajar.

Golang yang digunakan adalah versi **1.8.3**. Direkomendasikan menggunakan versi tersebut, atau versi lain minimal **1.4.2** ke atas.

Link untuk download installer golang: <https://golang.org/dl/>. Anda bisa langsung unduh dari URL tersebut lalu lakukan instalasi sendiri, atau bisa mengikuti petunjuk di bab ini.

### A.2.1. Instalasi Golang di Windows

1. Download terlebih dahulu installer-nya. Pilih sesuai jenis bit yang digunakan.
  - 32bit => [go1.8.3.windows-386.msi](#)
  - 64bit => [go1.8.3.windows-amd64.msi](#)
2. Setelah ter-download, jalankan installer, klik **next** sampai proses instalasi selesai. By default jika anda tidak merubah path pada saat instalasi, Golang akan terinstall di `C:\go`. Path tersebut secara otomatis didaftarkan dalam **path variable**.
3. Buka **Command Prompt / CMD**, eksekusi perintah untuk mengecek versi Golang.

```
$ go version
```

4. Jika output adalah sama dengan Golang yang ter-install, menandakan instalasi berhasil.

Sering terjadi command `go version` tidak bisa dijalankan meskipun instalasi sukses. Solusinya bisa dengan restart CMD (close CMD, lalu buka kembali). Setelah itu coba jalankan sekali lagi command tersebut.

### A.2.2. Instalasi Golang di Mac OS

Cara termudah instalasi Golang di **Mac OS** adalah menggunakan [homebrew](#).

1. Install terlebih dahulu Homebrew (jika belum ada), jalankan perintah tersebut di **terminal**.

```
$ ruby -e "$(curl -fsSL http://git.io/pv01)"
```

2. Install Golang menggunakan command `brew`.

```
$ brew install go
```

3. Tambahkan path ke environment variable.

```
$ echo "export PATH=$PATH:/usr/local/go/bin" >> ~/.bash_profile
$ source ~/.bash_profile
```

4. Jalankan command untuk mengecek versi Golang.

```
$ go version
```

5. Jika output adalah sama dengan Golang yang ter-install, menandakan instalasi berhasil.

### A.2.3. Instalasi Golang di Linux

1. Download archive berikut, pilih sesuai jenis bit komputer anda.

- o 32bit => [go1.8.3.linux-386.tar.gz](#)
- o 64bit => [go1.8.3.linux-amd64.tar.gz](#)

Download bisa dilakukan lewat CLI, menggunakan `wget` atau `curl`.

```
$ wget https://storage.googleapis.com/golang/go1.8.3.linux-amd64.tar.gz
```

2. Buka **terminal**, extract archive tersebut ke `/usr/local`.

```
$ tar -C /usr/local -xzf go1.8.3.linux-amd64.tar.gz
```

3. Setelah itu export path-nya, gunakan command di bawah ini.

```
$ echo "export PATH=$PATH:/usr/local/go/bin" >> ~/.bashrc  
$ source ~/.bashrc
```

4. Selanjutnya, eksekusi perintah berikut untuk mengetes apakah Golang sudah terinstal dengan benar.

```
$ go version
```

5. Jika output adalah sama dengan Golang yang ter-install, menandakan instalasi berhasil.

### A.2.4. Variabel `GOROOT`

By default, setelah proses instalasi Golang selesai, secara otomatis akan muncul environment variabel bernama `GOROOT`. Isi dari environment variabel ini adalah path/folder/lokasi dimana golang di install.

Sebagai contoh di Windows, ketika golang di-install di `c:\go`, maka path tersebut akan menjadi isi dari `GOROOT`.

## A.3. GOPATH Dan Workspace

Ada beberapa hal yang perlu disiapkan sebelum bisa masuk ke sesi pembuatan aplikasi menggunakan Golang, yaitu setup workspace untuk Project yang akan dibuat. Dan di bab ini kita akan belajar bagaimana caranya.

### A.3.1. Variabel GOPATH

**GOPATH** adalah variabel yang digunakan oleh Golang sebagai rujukan lokasi dimana semua folder project disimpan. Gopath berisikan 3 buah sub folder: `src`, `bin`, dan `pkg`.

Project di Golang **harus** ditempatkan dalam `$GOPATH/src`. Sebagai contoh anda ingin membuat project dengan nama `belajar`, maka **harus** dibuatkan sebuah folder dengan nama `belajar`, ditempatkan dalam `src` (`$GOPATH/src/belajar`).

Path separator yang digunakan sebagai contoh di buku ini adalah slash `/`. Khusus pengguna Windows, path separator adalah backslash `\`.

Ada pengecualian untuk *aturan project harus ditempatkan dalam workspace*. Jika project dikembangkan dengan menerapkan **Go Modules** maka tidak perlu ditaruh dalam workspace, lebih jelasnya akan dibahas secara mendetail di [bab A.60. Go Modules](#).

### A.3.2. Setup Workspace

Lokasi folder yang akan dijadikan sebagai workspace bisa ditentukan sendiri. Anda bisa menggunakan alamat folder mana saja, bebas, tapi jangan gunakan path dimana golang di-install (jangan sama dengan `GOROOT`). Lokasi tersebut harus didaftarkan dalam path variable dengan nama `GOPATH`. Sebagai contoh, penulis memilih path `$HOME/Documents/go`, maka saya daftarkan alamat tersebut. Caranya:

- Bagi pengguna **Windows**, tambahkan path folder tersebut ke **path variable** dengan nama `GOPATH`. Setelah variabel terdaftar, cek apakah path sudah terdaftar dengan benar.
 

Sering terjadi `GOPATH` tidak dikenali meskipun variabel sudah didaftarkan. Jika hal seperti ini terjadi, restart CMD, lalu coba lagi.
- Bagi pengguna Mac OS, export path ke `~/.bash_profile`. Untuk Linux, export ke `~/.bashrc`

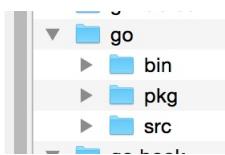
```
$ echo "export GOPATH=$HOME/Documents/go" >> ~/.bash_profile
$ source ~/.bash_profile
```

Cek apakah path sudah terdaftar dengan benar.

```
[novalagung:~ $ source ~/.bash_profile
[novalagung:~ $ echo $GOPATH
/Users/novalagung/Documents/go
novalagung:~ $ ]
```

Setelah `GOPATH` berhasil dikenali, perlu disiapkan 3 buah sub folder didalamnya, dengan kriteria sebagai berikut:

- Folder `src`, adalah path dimana project golang disimpan
- Folder `pkg`, berisi file hasil kompilasi
- Folder `bin`, berisi file executable hasil build



Struktur diatas merupakan struktur standar workspace Golang. Jadi pastikan penamaan dan hirarki folder adalah sama.

## A.4. Instalasi Editor

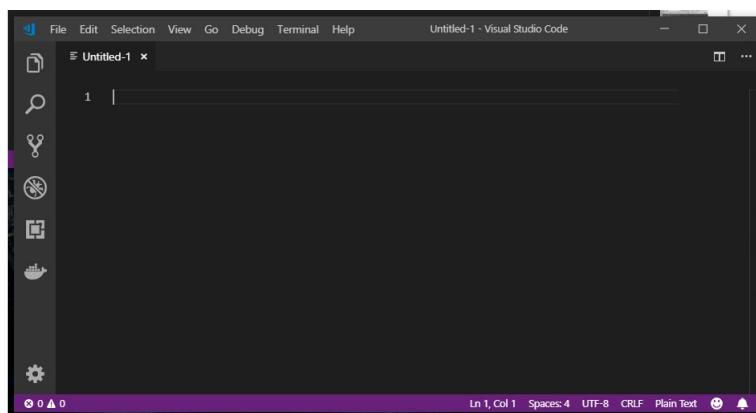
Proses pembuatan aplikasi menggunakan Golang akan lebih maksimal jika didukung oleh editor atau **IDE** yang pas. Ada cukup banyak pilihan bagus yang bisa dipertimbangkan, diantaranya: Brackets, JetBrains GoLand, Netbeans, Atom, Brackets, Visual Studio Code, Sublime Text, dan lainnya.

Penulis sarankan untuk memilih editor yang paling nyaman digunakan, preferensi masing-masing pastinya berbeda. Penulis sendiri sempat menggunakan Sublime Text 3, tapi sekarang pindah ke **Visual Studio Code**. Editor ini sangat ringan, mudah didapat, dan memiliki ekstensi yang bagus untuk bahasa Go. Jika pembaca ingin menggunakan editor yang sama, maka silakan melanjutkan guide berikut.

Di bab ini akan dijelaskan bagaimana cara instalasi editor Visual Studio Code.

### A.4.1. Instalasi Editor Visual Studio Code

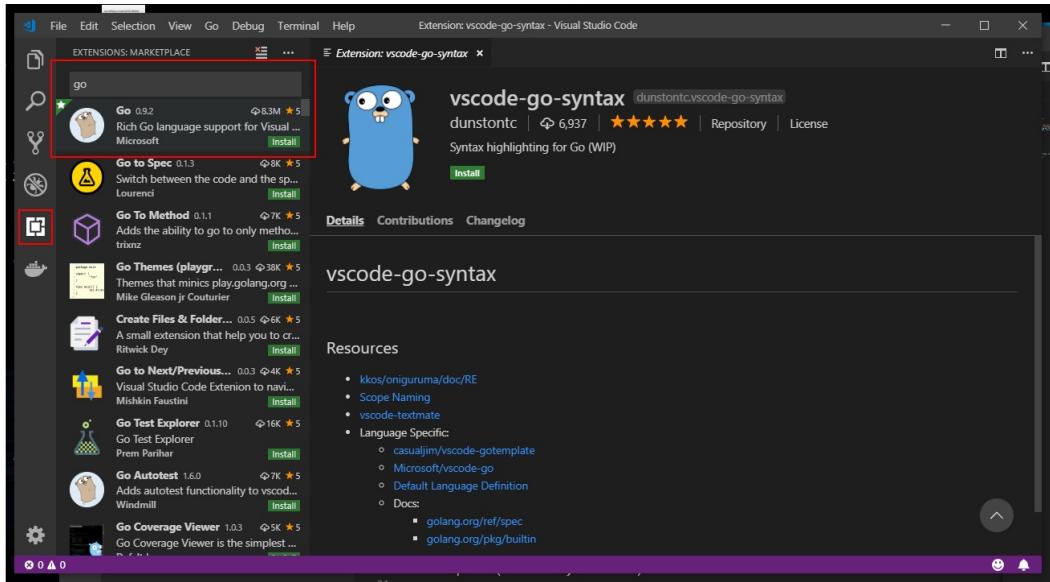
1. Download Visual Studio Code di <https://code.visualstudio.com/Download>, pilih sesuai dengan sistem operasi yang digunakan.
2. Jalankan installer.
3. Setelah selesai, jalankan aplikasi.



### A.4.2. Instalasi Extensi Go

Dengan meng-instal Go Extension, maka programming menggunakan bahasa ini akan menjadi sangat nyaman lewat VS Code. Banyak benefit yang didapat dengan meng-install ekstensi ini, beberapa diantaranya adalah integrasi dengan kompiler Go, auto lint on save, testing with coverage, fasilitas debugging with breakpoints, dan lainnya.

Cara instalasi ekstensi sendiri cukup mudah, klik `View -> Extension` atau klik ikon *Extension Marketplace* di sebelah kiri (silakan lihat gambar berikut, deretan button paling kiri yang dilingkari merah). Setelah itu ketikkan **Go** pada inputan search, silakan install ekstensi Go buatan Microsoft, biasanya muncul paling atas sendiri.



## A.5. Command

Pengembangan aplikasi Golang tak jauh dari hal-hal yang berbau CLI atau **Command Line Interface**. Proses kompilasi, testing, eksekusi program, semua dilakukan lewat command line.

Golang menyediakan command `go`, di bab ini kita akan belajar mengenai pemanfaatannya.

### A.5.1. Command `go run`

Command `go run` digunakan untuk eksekusi file program (file ber-ekstensi `.go`). Cara penggunaannya dengan menuliskan command tersebut diikut argumen nama file.

Berikut adalah contoh penerapan `go run` untuk eksekusi file program `bab5.go` yang tersimpan di path `$GOPATH/src/belajar-golang`.

```
$ cd $GOPATH/src/belajar-golang
$ go run bab5.go
```

```
[novalagung:~ $ cd $GOPATH/src/belajar-golang
[novalagung:belajar-golang $ go run bab5.go
Hello World
novalagung:belajar-golang $ ]
```

Command `go run` hanya bisa digunakan pada file yang package-nya adalah **main**. Lebih jelasnya dibahas pada bab selanjutnya (bab 6).

Jika ada banyak file yang ber-package `main`, dan file-file tersebut di-import di file utama, maka eksekusinya adalah dengan menyiapkan semua file sebagai argument `go run` (lebih jelasnya akan dibahas pada bab 25). Contohnya bisa dilihat pada kode berikut.

```
$ go run bab5.go library.go
```

```
[novalagung:belajar-golang $ go run bab5.go library.go
Hello World
novalagung:belajar-golang $ ]
```

Atau bisa dengan menggunakan `*.go`, tanpa tidak perlu menuliskan nama-nama file program yang ada.

```
$ go run *.go
```

### A.5.2. Command `go test`

Golang menyediakan package `testing`, berguna untuk keperluan unit testing. File yang akan di-test harus ber-suffix `_test.go`.

Berikut adalah contoh penggunaan command `go test` untuk testing file `bab5_test.go`.

```
$ go test bab5_test.go
```

```
[novalagung:belajar-golang $ go test bab5_test.go
ok      command-line-arguments 0.011s
novalagung:belajar-golang $ ]
```

### A.5.3. Command go build

Command ini digunakan untuk mengkompilasi file program.

Sebenarnya ketika eksekusi program menggunakan `go run`, terjadi proses kompilasi juga, file hasil kompilasi akan disimpan pada folder temporary untuk selanjutnya langsung dieksekusi.

Berbeda dengan `go build`, command ini menghasilkan file executable pada folder yang sedang aktif. Contohnya bisa dilihat pada kode berikut.

```
[novalagung:belajar-golang $ go build bab5.go
[novalagung:belajar-golang $ ./bab5
Hello World
novalagung:belajar-golang $ ]]
```

Pada contoh di atas, file `bab5.go` di-build, menghasilkan file baru pada folder yang sama, yaitu `bab5`, yang kemudian dieksekusi.

Pada pengguna Windows, file executable ber-ekstensi `.exe`.

### A.5.4. Command go install

Command `go install` memiliki fungsi yang sama dengan `go build`, hanya saja setelah proses kompilasi selesai, dilanjutkan ke proses instalasi program yang bersangkutan.

Target eksekusi harus berupa folder proyek (bukan file `.go`), dan path folder tersebut dituliskan relatif terhadap `$GOPATH/src`. Contoh:

```
$ go install github.com/novalagung/godong
```

`go install` menghasilkan output berbeda untuk package `main` dan non-main.

- Pada package **non-main**, menghasilkan file berekstensi `.a` tersimpan dalam folder `$GOPATH/pkg`.
- Pada package **main**, menghasilkan file **executable** tersimpan dalam folder `$GOPATH/bin`.

Berikut merupakan contoh penerapan `go install`.

```
[novalagung:godong $ go install github.com/novalagung/godong
[novalagung:godong $ go install github.com/novalagung/godong/godong_test
[novalagung:godong $ ls $GOPATH/pkg/darwin_amd64/github.com/novalagung/
godong          godong.a
[novalagung:godong $ ls $GOPATH/bin
godong_test
[novalagung:godong $ $GOPATH/bin/godong_test
route /dashboard/about-us
    -> Dashboard.Action_AboutUs
route /dashboard/data-analytic/get-data
    -> Dashboard.Action_DataAnalytic_GetData
route /dashboard/home
    -> Dashboard.Action_Home
[novalagung:godong $ ]]]]
```

Pada kode di atas bisa dilihat command `go install` dieksekusi 2 kali.

1. Pada package non-main, `github.com/novalagung/godong`. Hasil instalasi adalah file berekstensi `.a` tersimpan pada folder `$GOPATH/pkg`.
2. Pada package main, `github.com/novalagung/godong/godong_test`. Hasil instalasi adalah file executable tersimpan pada folder `$GOPATH/bin`.

### A.5.5. Command go get

Command ini berbeda dengan command-command yang sudah dibahas di atas. `go get` digunakan untuk men-download package. Sebagai contoh saya ingin men-download package **Mgo**.

```
$ go get gopkg.in/mgo.v2
$ ls $GOPATH/src/gopkg.in/mgo.v2
```

[novalagung:src \$ go get gopkg.in/mgo.v2	export_test.go	session.go
[novalagung:src \$ ls \$GOPATH/src/gopkg.in/mgo.v2	gridfs.go	session_test.go
LICENSE	gridfs_test.go	socket.go
Makefile	internal	stats.go
README.md	log.go	suite_test.go
auth.go	queue.go	syscall_test.go
auth_test.go	queue_test.go	syscall_windows_test.go
bson	raceoff.go	testdb
bulk.go	raceon.go	testserver
bulk_test.go	saslimpl.go	txn
cluster.go	saslstub.go	
cluster_test.go	server.go	
dbtest		
doc.go		
novalagung:src \$		

[gopkg.in/mgo.v2](http://gopkg.in/mgo.v2) adalah URL package mgo. Package yang sudah ter-download tersimpan dalam `$GOPATH/src`, dengan struktur folder sesuai dengan URL package-nya. Sebagai contoh, package MGO di atas tersimpan di `$GOPATH/src/gopkg.in/mgo.v2`.

## A.6. Program Pertama: Hello World

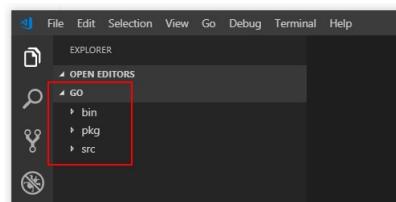
Semua persiapan sudah selesai, saatnya mulai masuk pada sesi pembuatan program. Program pertama yang akan kita buat adalah aplikasi kecil untuk memunculkan tulisan **Hello World**.

Di bab ini akan dijelaskan secara bertahap dari awal. Mulai pembuatan project, pembuatan file program, sesi penulisan kode (coding), hingga eksekusi aplikasi.

### A.6.1. Load GOPATH Ke Editor

Hal pertama yang perlu dilakukan, adalah me-load atau memunculkan folder `GOPATH` di editor. Dengan begitu proyek-proyek Golang akan lebih mudah di-maintain. Caranya:

1. Buka editor yang digunakan.
2. Cari menu untuk menambahkan projek, lalu pilih folder `GOPATH`. Untuk beberapa jenis editor bisa cukup dengan klik-drag folder tersebut ke editor.

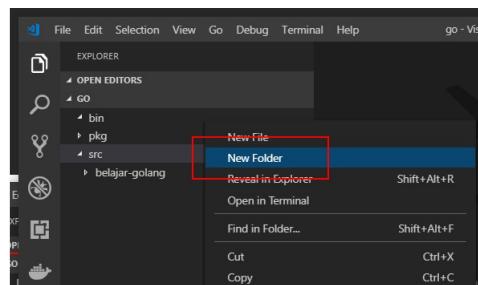


Nama variabel di sistem operasi non-Windows diawali dengan tanda dollar `$`, sebagai contoh `$GOPATH`. Sedangkan di Windows, nama variabel diapit karakter persen `%`, contohnya seperti `%GOPATH%`.

### A.6.2. Menyiapkan Folder Project

Selanjutnya, buat project folder baru dalam `$GOPATH/src`, dengan nama folder bebas (boleh menggunakan nama `belajar-golang` atau lainnya). Agar lebih praktis, buat folder tersebut lewat editor yang digunakan. Berikut adalah caranya.

1. Klik kanan di folder `src`.
2. Klik **New Folder**, di bagian bawah akan muncul inputan kecil **Folder Name**.
3. Ketikkan nama folder, **belajar-golang**, lalu enter.



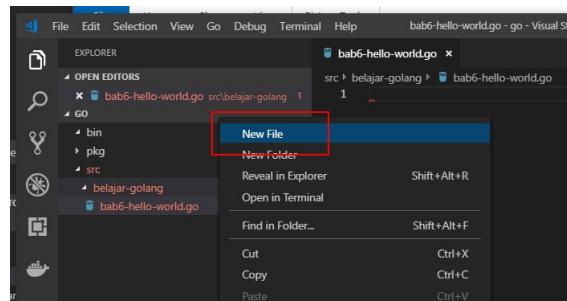
### A.6.3. Menyiapkan File Program

File program disini maksudnya adalah file yang berisikan kode program Golang, yang berekstensi `.go`.

Di dalam project yang telah dibuat (`$GOPATH/src/belajar-golang/`), siapkan sebuah file dengan nama bebas, yang jelas harus ber-ekstensi `.go` (Pada contoh ini saya menggunakan nama file `bab6-hello-world.go`).

Pembuatan file program juga akan dilakukan lewat editor. Caranya silakan ikut petunjuk berikut.

1. Klik kanan di folder `belajar-golang`.
2. Klik **New File**, di bagian bawah akan muncul inputan kecil **File Name**.
3. Ketikkan nama file, **belajar-golang**, lalu enter.



## A.6.4. Program Pertama: Hello Word

Setelah project folder dan file program sudah siap, saatnya untuk **coding**.

Dibawah ini merupakan contoh kode program sederhana untuk memunculkan text atau tulisan "**hello world**" ke layar output (command line).

Silakan salin kode berikut ke file program yang telah dibuat. Sebisa mungkin jangan copy paste. Biasakan untuk menulis dari awal, agar cepat terbiasa dan familiar dengan Golang.

```
package main

import "fmt"

func main() {
    fmt.Println("hello world")
}
```

Setelah kode disalin, buka terminal (atau CMD bagi pengguna Windows), lalu masuk ke direktori proyek menggunakan perintah `cd`.

- Windows

```
$ cd %GOPATH%\src\belajar-golang
```

- Non-Windows

```
$ cd $GOPATH/src/belajar-golang
```

Jalankan program dengan perintah `go run`.

```
$ go run bab6-hello-world.go
```

Hasilnya, muncul tulisan **hello world** di layar console.

```

File Edit Selection View Go Debug Terminal Help
bab6-hello-world.go - go - Visual Studio Code
EXPLORER
OPEN EDITORS
bab6-hello-world.go src\belajar-golang
GO
bin
pkg
src
belajar-golang
bab6-hello-world.go
Command Prompt
C:\Users\novalagung\Desktop\backup\temp\go\src\belajar-golang>go run bab6-hello-world.go
hello world

```

Selamat! Anda telah berhasil membuat program menggunakan Golang!

Meski kode program di atas sangat sederhana, mungkin akan muncul beberapa pertanyaan di benak. Di bawah ini merupakan detail penjelasan kode di atas.

## A.6.5. Penggunaan Keyword `package`

Setiap file program harus memiliki package. Setiap project harus ada satu file dengan package bernama `main`. File yang ber-package `main`, akan di eksekusi pertama kali ketika program di jalankan.

Cara menentukan package dengan menggunakan keyword `package`, berikut adalah contoh penulisannya.

```

package <nama-package>
package main

```

## A.6.6. Penggunaan Keyword `import`

Keyword `import` digunakan untuk meng-include atau memasukan package lain kedalam file program, agar isi package yang di-include bisa dimanfaatkan.

Package `fmt` merupakan salah satu package yang disediakan oleh Golang, berisikan banyak fungsi untuk keperluan I/O yang berhubungan dengan text.

Skema penulisan keyword `import` bisa dilihat pada contoh berikut.

```

import "<nama-package>" 
import "fmt"

```

## A.6.7. Penggunaan Fungsi `main()`

Dalam sebuah proyek harus ada file program yang berisikan sebuah fungsi bernama `main()`. Fungsi tersebut harus berada dalam package yang juga bernama `main`. Fungsi `main()` adalah yang dipanggil pertama kali pada saat eksekusi program. Contoh penulisan fungsi `main` :

```

func main() {
}

```

## A.6.8. Penggunaan Fungsi `fmt.Println()`

Fungsi `fmt.Println()` digunakan untuk memunculkan text ke layar (pada konteks ini, terminal atau CMD). Di program pertama yang telah kita buat, fungsi ini memunculkan tulisan **Hello World**.

Skema penulisan keyword `fmt.Println()` bisa dilihat pada contoh berikut.

```
fmt.Println("<isi-pesan>")  
fmt.Println("hello world")
```

Fungsi `fmt.Println()` berada dalam package `fmt`, maka untuk menggunakannya perlu package tersebut untuk di-import terlebih dahulu.

Fungsi `fmt.Println()` dapat menampung parameter yang tidak terbatas jumlahnya. Semua data parameter akan dimunculkan dengan pemisah tanda spasi.

```
fmt.Println("hello", "world!", "how", "are", "you")
```

Outputnya: **hello world! how are you.**

## A.7. Komentar

Komentar biasa dimanfaatkan untuk menyisipkan catatan pada kode program, menulis penjelasan atau deskripsi mengenai suatu blok kode, atau bisa juga digunakan untuk me-remark kode (men-non-aktifkan kode yg tidak digunakan). Komentar akan diabaikan ketika kompilasi maupun eksekusi program.

Ada 2 jenis komentar di Golang, inline & multiline. Di bab akan dijelaskan tentang penerapan dan perbedaan kedua jenis komentar tersebut.

### A.7.1. Komentar Inline

Penulisan komentar jenis ini di awali dengan tanda **double slash** ( // ) lalu diikuti pesan komentarnya. Komentar inline hanya berlaku untuk satu baris pesan saja. Jika pesan komentar lebih dari satu baris, maka tanda // harus ditulis lagi di baris selanjutnya.

```
package main

import "fmt"

func main() {
    // komentar kode
    // menampilkan pesan hello world
    fmt.Println("hello world")

    // fmt.Println("baris ini tidak akan di eksekusi")
}
```

Mari kita praktikan kode di atas. Siapkan file program baru dalam project folder `belajar-golang` dengan nama bebas. Isi dengan kode di atas, lalu jalankan.

```
[novalagung:belajar-golang $ go run bab7.go
hello world
novalagung:belajar-golang $ ]
```

Hasilnya hanya tulisan **hello world** saja yang muncul di layar, karena semua yang di awali tanda double slash // diabaikan oleh compiler.

### A.7.2. Komentar Multiline

Komentar yang cukup panjang akan lebih rapi jika ditulis menggunakan teknik komentar multiline. Ciri dari komentar jenis ini adalah penulisannya diawali dengan tanda /\* dan diakhiri \*/ .

```
/*
komentar kode
menampilkan pesan hello world
*/
fmt.Println("hello world")

// fmt.Println("baris ini tidak akan di eksekusi")
```

Sifat komentar ini sama seperti komentar inline, yaitu sama-sama diabaikan oleh compiler.



## A.8. Variabel

Golang mengadopsi dua jenis penulisan variabel, yang dituliskan tipe data-nya dan yang tidak. Kedua cara tersebut valid dan tujuannya sama, pembedanya hanya cara penulisannya saja.

Pada bab ini akan dikupas tuntas tentang macam-macam cara deklarasi variabel.

### A.8.1. Deklarasi Variabel Dengan Tipe Data

Golang memiliki aturan cukup ketat dalam hal penulisan variabel. Ketika deklarasi, tipe data yg digunakan harus dituliskan juga. Istilah lain dari konsep ini adalah **manifest typing**.

Berikut adalah contoh cara pembuatan variabel yang tipe datanya harus ditulis.

```
package main

import "fmt"

func main() {
    var firstName string = "john"

    var lastName string
    lastName = "wick"

    fmt.Printf("halo %s %s!\n", firstName, lastName)
}
```

Keyword `var` di atas digunakan untuk deklarasi variabel, contohnya bisa dilihat pada `firstName` dan `lastName`.

Nilai variabel `firstName` diisi langsung ketika deklarasi, berbeda dibanding `lastName` yang nilainya diisi setelah baris kode deklarasi, hal seperti ini diperbolehkan di Golang.

```
[novalagung:belajar-golang $ go run bab8.go
halo john wick!
novalagung:belajar-golang $ ]
```

### A.8.2. Deklarasi Variabel Menggunakan Keyword `var`

Pada kode di atas bisa dilihat bagaimana sebuah variabel dideklarasikan dan di-set nilainya. Keyword `var` digunakan untuk membuat variabel baru.

Skema penggunaan keyword `var`:

```
var <nama-variabel> <tipe-data>
var <nama-variabel> <tipe-data> = <nilai>
```

Contoh:

```
var lastName string
var firstName string = "john"
```

Nilai variabel bisa di-isi langsung pada saat deklarasi variabel.

### A.8.3. Penggunaan Fungsi `fmt.Printf()`

Fungsi ini digunakan untuk menampilkan output dalam bentuk tertentu. Kegunaannya sama seperti fungsi `fmt.Println()`, hanya saja struktur outputnya didefinisikan di awal.

Perhatikan bagian `"halo %s %s!\n"`, karakter `%s` disitu akan diganti dengan data `string` yang berada di parameter ke-2, ke-3, dan seterusnya.

Ketiga baris kode di bawah ini menghasilkan output yang sama, meskipun cara penulisannya berbeda.

```
fmt.Printf("halo john wick!\n")
fmt.Printf("halo %s %s!\n", firstName, lastName)
fmt.Println("halo", firstName, lastName + "!")
```

Tanda plus (`+`) jika ditempatkan di antara string, fungsinya adalah untuk penggabungan string (istilah lainnya: concatenation).

Fungsi `fmt.Printf()` tidak menghasilkan baris baru di akhir text, oleh karena itu digunakanlah literal `\n` untuk memunculkan baris baru di akhir. Hal ini sangat berbeda jika dibandingkan dengan fungsi `fmt.Println()` yang secara otomatis menghasilkan new line (baris baru) di akhir.

### A.8.4. Deklarasi Variabel Tanpa Tipe Data

Selain **manifest typing**, GoLang juga mengadopsi metode **type inference**, yaitu metode deklarasi variabel yang tipe data-nya ditentukan oleh tipe data nilainya, cara kontradiktif jika dibandingkan dengan cara pertama. Dengan metode jenis ini, keyword `var` dan tipe data tidak perlu dituliskan.

```
var firstName string = "john"
lastName := "wick"

fmt.Printf("halo %s %s!\n", firstName, lastName)
```

Variabel `lastName` dideklarasikan dengan menggunakan metode type inference. Penandanya tipe data tidak dituliskan pada saat deklarasi. Pada penggunaan metode ini, operand `=` harus diganti dengan `:=` dan keyword `var` dihilangkan.

Tipe data `lastName` secara otomatis akan ditentukan menyesuaikan value atau nilai-nya. Jika nilainya adalah berupa `string` maka tipe data variabel adalah `string`. Pada contoh di atas, nilainya adalah string `"wick"`.

Diperbolehkan untuk tetap menggunakan keyword `var` pada saat deklarasi meskipun tanpa menuliskan tipe data, dengan ketentuan tidak menggunakan tanda `:=`, melainkan tetap menggunakan `=`.

```
// menggunakan var, tanpa tipe data, menggunakan perantara "="
var firstName = "john"

// tanpa var, tanpa tipe data, menggunakan perantara ":="
lastName := "wick"
```

Kedua deklarasi di atas maksudnya sama. Silakan pilih yang nyaman di hati.

Tanda `:=` hanya digunakan sekali di awal pada saat deklarasi. Untuk assignment nilai selanjutnya harus menggunakan tanda `=`, contoh:

```
lastName := "wick"
lastName = "ethan"
lastName = "bourne"
```

Perlu diketahui, deklarasi menggunakan `:=` hanya bisa dilakukan di dalam fungsi, tidak bisa digunakan di luar fungsi.

## A.8.5. Deklarasi Multi Variabel

Golang mendukung metode deklarasi banyak variabel secara bersamaan, caranya dengan menuliskan variabel-variabelnya dengan pembatas tanda koma ( , ). Untuk pengisian nilainya-pun diperbolehkan secara bersamaan.

```
var first, second, third string
first, second, third = "satu", "dua", "tiga"
```

Pengisian nilai juga bisa dilakukan bersamaan pada saat deklarasi. Caranya dengan menuliskan nilai masing-masing variabel berurutan sesuai variabelnya dengan pembatas koma ( , ).

```
var fourth, fifth, sixth string = "empat", "lima", "enam"
```

Kalau ingin lebih ringkas:

```
seventh, eighth, ninth := "tujuh", "delapan", "sembilan"
```

Dengan menggunakan teknik type inference, deklarasi multi variabel bisa dilakukan untuk variabel-variabel yang tipe data satu sama lainnya berbeda.

```
one, isFriday, twoPointTwo, say := 1, true, 2.2, "hello"
```

Istimewa bukan? Istimewa sekali.

## A.8.6. Variabel Underscore `_`

Golang memiliki aturan unik yang jarang dimiliki bahasa lain, yaitu tidak boleh ada satupun variabel yang menganggur. Artinya, semua variabel yang dideklarasikan harus digunakan. Jika ada variabel yang tidak digunakan tapi dideklarasikan, error akan muncul dan program tidak bisa di-run ataupun di-compile.

```
[novalagung:belajar-golang $ go run bab8.go
# command-line-arguments
./bab8.go:6: name declared and not used
novalagung:belajar-golang $ ]
```

Underscore (`_`) adalah predefined variabel yang bisa dimanfaatkan untuk menampung nilai yang tidak dipakai. Bisa dibilang variabel ini merupakan keranjang sampah.

```
_ = "belajar Golang"
_ = "Golang itu mudah"
name, _ := "john", "wick"
```

Pada contoh di atas, variabel `name` akan berisikan text `john`, sedang nilai `wick` ditampung oleh variabel underscore, menandakan bahwa nilai tersebut tidak akan digunakan.

Variabel underscore adalah predefined, jadi tidak perlu menggunakan `:=` untuk pengisian nilai, cukup dengan `=` saja. Namun khusus untuk pengisian nilai multi variabel yang dilakukan dengan metode type inference, boleh didalamnya terdapat variabel underscore.

Biasanya variabel underscore sering dimanfaatkan untuk menampung nilai balik fungsi yang tidak digunakan.

Perlu diketahui, bahwa isi variabel underscore tidak dapat ditampilkan. Data yang sudah masuk variabel tersebut akan hilang. Ibarat blackhole, sekali masuk, tidak akan bisa keluar

## A.8.7. Deklarasi Variabel Menggunakan Keyword `new`

Keyword `new` digunakan untuk mencetak data **pointer** dengan tipe data tertentu. Nilai data default-nya akan menyesuaikan tipe datanya.

```
name := new(string)

fmt.Println(name)    // 0x20818a220
fmt.Println(*name)  // ""
```

Variabel `name` menampung data bertipe **pointer string**. Jika ditampilkan yang muncul bukanlah nilainya melainkan alamat memori nilai tersebut (dalam bentuk notasi heksadesimal). Untuk menampilkan nilai aslinya, variabel tersebut perlu di-**dereference** terlebih dahulu, menggunakan tanda asterisk (`*`).

Mungkin untuk sekarang banyak yang akan bingung tentang apa itu pointer, namun tak apa, karena nantinya di bab 22 akan dikupas habis topik pointer dan dereference.

## A.8.8. Deklarasi Variabel Menggunakan Keyword `make`

Keyword ini hanya bisa digunakan untuk pembuatan beberapa jenis variabel saja, yaitu:

- channel
- slice
- map

Dan lagi, mungkin banyak yang akan bingung. Ketika sudah masuk ke pembahasan masing-masing point tersebut, akan terlihat apa kegunaan dari keyword `make` ini.

## A.9. Tipe Data

Golang mengenal beberapa jenis tipe data, diantaranya adalah tipe data numerik (desimal & non-desimal), string, dan boolean.

Di bab-bab sebelumnya secara tak sadar kita sudah mengaplikasikan beberapa tipe data, seperti `string` dan tipe numerik `int`.

Bab ini menjelaskan beberapa macam tipe data standar yang disediakan oleh Golang, beserta cara penggunaannya.

### A.9.1. Tipe Data Numerik Non-Desimal

Tipe data numerik non-desimal atau **non floating point** di Golang ada beberapa jenis. Secara umum ada 2 tipe data kategori ini yang perlu diketahui.

- `uint`, tipe data untuk bilangan cacah (bilangan positif).
- `int`, tipe data untuk bilangan bulat (bilangan negatif dan positif).

Kedua tipe data di atas kemudian dibagi lagi menjadi beberapa jenis, dengan pembagian berdasarkan lebar cakupan nilainya, detailnya bisa dilihat di tabel berikut.

Tipe data	Cakupan bilangan
<code>uint8</code>	$0 \leftrightarrow 255$
<code>uint16</code>	$0 \leftrightarrow 65535$
<code>uint32</code>	$0 \leftrightarrow 4294967295$
<code>uint64</code>	$0 \leftrightarrow 18446744073709551615$
<code>uint</code>	sama dengan <code>uint32</code> atau <code>uint64</code> (tergantung nilai)
<code>byte</code>	sama dengan <code>uint8</code>
<code>int8</code>	$-128 \leftrightarrow 127$
<code>int16</code>	$-32768 \leftrightarrow 32767$
<code>int32</code>	$-2147483648 \leftrightarrow 2147483647$
<code>int64</code>	$-9223372036854775808 \leftrightarrow 9223372036854775807$
<code>int</code>	sama dengan <code>int32</code> atau <code>int64</code> (tergantung nilai)
<code>rune</code>	sama dengan <code>int32</code>

Dianjurkan untuk tidak sembarangan dalam menentukan tipe data variabel, sebisa mungkin tipe yang dipilih harus disesuaikan dengan nilainya, karena efeknya adalah ke alokasi memori variabel. Pemilihan tipe data yang tepat akan membuat pemakaian memori lebih optimal, tidak berlebihan.

```
var positiveNumber uint8 = 89
var negativeNumber = -1243423644

fmt.Printf("bilangan positif: %d\n", positiveNumber)
fmt.Printf("bilangan negatif: %d\n", negativeNumber)
```

Variabel `positiveNumber` bertipe `uint8` dengan nilai awal `89`. Sedangkan variabel `negativeNumber` dideklarasikan dengan nilai awal `-1243423644`. Compiler secara cerdas akan menentukan tipe data variabel tersebut sebagai `int32` (karena angka tersebut masuk ke cakupan tipe data `int32`).

Template `%d` pada `fmt.Printf()` digunakan untuk memformat data numerik non-desimal.

## A.9.2. Tipe Data Numerik Desimal

Tipe data numerik desimal yang perlu diketahui ada 2, `float32` dan `float64`. Perbedaan kedua tipe data tersebut berada di lebar cakupan nilai desimal yang bisa ditampung. Untuk lebih jelasnya bisa merujuk ke spesifikasi [IEEE-754 32-bit floating-point numbers](#).

```
var decimalNumber = 2.62

fmt.Printf("bilangan desimal: %f\n", decimalNumber)
fmt.Printf("bilangan desimal: %.3f\n", decimalNumber)
```

Pada kode di atas, variabel `decimalNumber` akan memiliki tipe data `float32`, karena nilainya berada di cakupan tipe data tersebut.

```
[novalagung:belajar-golang $ go run bab9.go
bilangan desimal: 2.620000
bilangan desimal: 2.620
novalagung:belajar-golang $ ]
```

Template `%f` digunakan untuk memformat data numerik desimal menjadi string. Digit desimal yang akan dihasilkan adalah **6 digit**. Pada contoh di atas, hasil format variabel `decimalNumber` adalah `2.620000`. Jumlah digit yang muncul bisa dikontrol menggunakan `%.nf`, tinggal ganti `n` dengan angka yang diinginkan. Contoh: `%.3f` maka akan menghasilkan 3 digit desimal, `%.10f` maka akan menghasilkan 10 digit desimal.

## A.9.3. Tipe Data `bool` (Boolean)

Tipe data `bool` berisikan hanya 2 varian nilai, `true` dan `false`. Tipe data ini biasa dimanfaatkan dalam seleksi kondisi dan perulangan (yang nantinya akan kita bahas pada bab 12 dan bab 13).

```
var exist bool = true
fmt.Printf("exist? %t \n", exist)
```

Gunakan `%t` untuk memformat data `bool` menggunakan fungsi `fmt.Printf()`.

## A.9.4. Tipe Data `string`

Ciri khas dari tipe data string adalah nilainya di apit oleh tanda *quote* atau petik dua ( " ). Contoh penerapannya:

```
var message string = "Halo"
fmt.Printf("message: %s \n", message)
```

Selain menggunakan tanda quote, deklarasi string juga bisa dengan tanda *grave accent/backticks* ( ` ), tanda ini terletak di sebelah kiri tombol 1. Keistimewaan string yang dideklarasikan menggunakan backtics adalah membuat semua karakter didalamnya **tidak di escape**, termasuk `\n`, tanda petik dua dan tanda petik satu, baris baru, dan lainnya. Semua akan terdeteksi sebagai string.

```
var message = `Nama saya "John Wick".`
```

```
Salam kenal.  
Mari belajar "Golang".`  
  
fmt.Println(message)
```

Ketika dijalankan, output akan muncul sama persis sesuai nilai variabel `message` di atas. Tanda petik dua akan muncul, baris baru juga muncul, sama persis.

```
[novalagung:belajar-golang $ go run bab9.go  
Nama saya "John Wick".  
Salam kenal.  
Mari belajar "Golang".  
novalagung:belajar-golang $ ]
```

## A.9.5. Nilai `nil` Dan Nilai Default Tipe Data

`nil` bukan merupakan tipe data, melainkan sebuah nilai. Variabel yang isi nilainya `nil` berarti memiliki nilai kosong.

Se semua tipe data yang sudah dibahas di atas memiliki nilai default. Artinya meskipun variabel dideklarasikan dengan tanpa nilai awal, akan ada nilai default-nya.

- Nilai default `string` adalah `""` (string kosong).
- Nilai default `bool` adalah `false`.
- Nilai default tipe numerik non-desimal adalah `0`.
- Nilai default tipe numerik desimal adalah `0.0`.

`nil` adalah nilai kosong, benar-benar kosong. `nil` tidak bisa digunakan pada tipe data yang sudah dibahas di atas, karena kesemuanya akan memiliki nilai default pada saat deklarasi. Ada beberapa tipe data yang bisa di-set nilainya dengan `nil`, diantaranya:

- pointer
- tipe data fungsi
- slice
- `map`
- `channel`
- interface kosong atau `interface{}`

Nantinya kita akan sering bertemu dengan `nil` setelah masuk pada pembahasan bab-bab tersebut.

## A.10. Konstanta

Konstanta adalah jenis variabel yang nilainya tidak bisa diubah. Inisialisasi nilai hanya dilakukan sekali di awal, setelahnya tidak bisa diubah nilainya.

### A.10.1. Penggunaan Konstanta

Data seperti `pi` ( $22/7$ ), kecepatan cahaya (299.792.458 m/s), adalah contoh data yang tepat jika dideklarasikan sebagai konstanta daripada variabel, karena nilainya sudah pasti dan tidak berubah.

Cara penerapan konstanta sama seperti deklarasi variabel biasa, selebihnya tinggal ganti keyword `var` dengan `const`.

```
const firstName string = "john"
fmt.Println("halo ", firstName, "!\\n")
```

Teknik type inference bisa diterapkan pada konstanta, caranya yaitu cukup dengan menghilangkan tipe data pada saat deklarasi.

```
const lastName = "wick"
fmt.Println("nice to meet you ", lastName, "!\\n")
```

### A.10.2. Penggunaan Fungsi `fmt.Println()`

Fungsi ini memiliki peran yang sama seperti fungsi `fmt.Println()`, pembedanya fungsi `fmt.Print()` tidak menghasilkan baris baru di akhir outputnya.

Perbedaan lainnya adalah, nilai pada parameter-parameter yang dimasukkan ke fungsi tersebut digabungkan tanpa pemisah. Tidak seperti pada fungsi `fmt.Println()` yang nilai paremeternya digabung menggunakan penghubung spasi.

```
fmt.Println("john wick")
fmt.Println("john", "wick")

fmt.Print("john wick\\n")
fmt.Print("john ", "wick\\n")
fmt.Print("john", " ", "wick\\n")
```

Kode di atas menunjukkan perbedaan antara `fmt.Println()` dan `fmt.Print()`. Output yang dihasilkan oleh 5 statement di atas adalah sama, meski cara yang digunakan berbeda.

Bila menggunakan `fmt.Println()` tidak perlu menambahkan spasi di tiap kata, karena fungsi tersebut akan secara otomatis menambahkannya di sela-sela nilai. Berbeda dengan `fmt.Print()`, perlu ditambahkan spasi, karena fungsi ini tidak menambahkan spasi di sela-sela nilai parameter yang digabungkan.

## A.11. Operator

Bab ini membahas mengenai macam operator yang bisa digunakan di Golang. Secara umum operator dibagi menjadi 3 kategori: operator aritmatika, perbandingan, dan logika.

### A.11.1. Operator Aritmatika

Operator aritmatika adalah operator yang digunakan untuk operasi yang sifatnya perhitungan. Golang mendukung beberapa operator aritmatika standar, list-nya bisa dilihat di tabel berikut.

Tanda	Penjelasan
+	penjumlahan
-	pengurangan
*	perkalian
/	pembagian
%	modulus / sisa hasil pembagian

Contoh penggunaan:

```
var value = (((2 + 6) % 3) * 4 - 2) / 3
```

### A.11.2. Operator Perbandingan

Operator perbandingan digunakan untuk menentukan kebenaran suatu kondisi. Hasilnya berupa nilai boolean, `true` atau `false`.

Tabel di bawah ini berisikan operator perbandingan yang bisa digunakan di Golang.

Tanda	Penjelasan
==	apakah nilai kiri <b>sama dengan</b> nilai kanan
!=	apakah nilai kiri <b>tidak sama dengan</b> nilai kanan
<	apakah nilai kiri <b>lebih kecil daripada</b> nilai kanan
<=	apakah nilai kiri <b>lebih kecil atau sama dengan</b> nilai kanan
>	apakah nilai kiri <b>lebih besar dari</b> nilai kanan
>=	apakah nilai kiri <b>lebih besar atau sama dengan</b> nilai kanan

Contoh penggunaan:

```
var value = (((2 + 6) % 3) * 4 - 2) / 3
var isEqual = (value == 2)

fmt.Printf("nilai %d (%t) \n", value, isEqual)
```

Pada kode di atas, terdapat statement operasi aritmatika yang hasilnya ditampung oleh variabel `value`. Selanjutnya, variabel tersebut tersebut dibandingkan dengan angka `2` untuk dicek apakah nilainya sama. Jika iya, maka hasilnya adalah `true`, jika tidak maka `false`. Nilai hasil operasi perbandingan tersebut kemudian disimpan dalam variabel `isEqual`.

```
[novalagung:belajar-golang $ go run bab11.go
 nilai 2 (true)
 novalagung:belajar-golang $ ]
```

Untuk memunculkan nilai `bool` menggunakan `fmt.Printf()`, bisa gunakan layout format `%t`.

### A.11.3. Operator Logika

Operator ini digunakan untuk mencari benar tidaknya kombinasi data bertipe `bool` (bisa berupa variabel bertipe `bool`, atau hasil dari operator perbandingan).

Beberapa operator logika standar yang bisa digunakan:

Tanda	Penjelasan
<code>&amp;&amp;</code>	kiri <b>dan</b> kanan
<code>  </code>	kiri <b>atau</b> kanan
<code>!</code>	negasi / nilai kebalikan

Contoh penggunaan:

```
var left = false
var right = true

var leftAndRight = left && right
fmt.Printf("left && right \t(%t) \n", leftAndRight)

var leftOrRight = left || right
fmt.Printf("left || right \t(%t) \n", leftOrRight)

var leftReverse = !left
fmt.Printf("!left \t\t(%t) \n", leftReverse)
```

Hasil dari operator logika sama dengan hasil dari operator perbandingan, yaitu berupa nilai boolean.

```
[novalagung:belajar-golang $ go run bab11.go
 left && right  (false)
 left || right  (true)
 !left          (true)
 novalagung:belajar-golang $ ]
```

Berikut penjelasan statemen operator logika pada kode di atas.

- `leftAndRight` bernilai `false`, karena hasil dari `false dan true` adalah `false`.
- `leftOrRight` bernilai `true`, karena hasil dari `false atau true` adalah `true`.
- `leftReverse` bernilai `true`, karena **negasi** (atau lawan dari) `false` adalah `true`.

Template `\t` digunakan untuk menambahkan indent tabulasi. Biasa dimanfaatkan untuk merapikan tampilan output pada console.



## A.12. Seleksi Kondisi

Seleksi kondisi digunakan untuk mengontrol alur program. Analoginya mirip seperti fungsi rambu lalu lintas di jalan raya. Kapan kendaraan diperbolehkan melaju dan kapan harus berhenti diatur oleh rambu tersebut. Sama seperti pada seleksi kondisi, kapan sebuah blok kode akan dieksekusi juga dikontrol.

Yang dijadikan acuan oleh seleksi kondisi adalah nilai bertipe `bool`, bisa berasal dari variabel, ataupun hasil operasi perbandingan. Nilai tersebut menentukan blok kode mana yang akan dieksekusi.

Golang memiliki 2 macam keyword untuk seleksi kondisi, yaitu `if else` dan `switch`. Di bab ini kita akan mempelajarinya satu-persatu.

Golang tidak mendukung seleksi kondisi menggunakan `ternary`.

Statement seperti: `var data = (isExist ? "ada" : "tidak ada")` adalah invalid dan menghasilkan error.

### A.12.1. Seleksi Kondisi Menggunakan Keyword `if` , `else if` , & `else`

Cara penerapan `if-else` di Golang sama seperti pada bahasa pemrograman lain. Yang membedakan hanya tanda kurungnya (*parentheses*), di Golang tidak perlu ditulis. Kode berikut merupakan contoh penerapan seleksi kondisi `if else`, dengan jumlah kondisi 4 buah.

```
var point = 8

if point == 10 {
    fmt.Println("lulus dengan nilai sempurna")
} else if point > 5 {
    fmt.Println("lulus")
} else if point == 4 {
    fmt.Println("hampir lulus")
} else {
    fmt.Printf("tidak lulus. nilai anda %d\n", point)
}
```

Dari keempat kondisi di atas, yang terpenuhi adalah `if point > 5`, karena nilai variabel `point` memang lebih besar dari `5`. Maka blok kode tepat dibawah kondisi tersebut akan dieksekusi (blok kode ditandai kurung kurawal buka dan tutup), hasilnya text `"lulus"` muncul sebagai output.

```
[novalagung:belajar-golang $ go run bab12.go
lulus
novalagung:belajar-golang $ ]
```

Skema `if else` Golang sama seperti pada pemrograman umumnya. Yaitu di awal seleksi kondisi menggunakan `if`, dan ketika kondisinya tidak terpenuhi akan menuju ke `else` (jika ada). Ketika ada banyak kondisi, gunakan `else if`.

Di bahasa pemrograman lain, ketika ada seleksi kondisi yang isi blok-nya hanya 1 baris saja, kurung kurawal boleh tidak dituliskan. Berbeda dengan aturan di Golang, kurung kurawal harus tetap dituliskan meski isinya hanya 1 blok statement.

### A.12.2. Variabel Temporary Pada `if` - `else`

Variabel temporary adalah variabel yang hanya bisa digunakan pada blok seleksi kondisi dimana ia ditempatkan saja. Penggunaan variabel ini membawa beberapa manfaat, antara lain:

- Scope atau cakupan variabel jelas, hanya bisa digunakan pada blok seleksi kondisi itu saja
- Kode menjadi lebih rapi
- Ketika nilai variabel tersebut didapat dari sebuah komputasi, perhitungan tidak perlu dilakukan di dalam blok masing-masing kondisi.

```
var point = 8840.0

if percent := point / 100; percent >= 100 {
    fmt.Printf("%.1f% perfect!\n", percent, "%")
} else if percent >= 70 {
    fmt.Printf("%.1f% good\n", percent, "%")
} else {
    fmt.Printf("%.1f% not bad\n", percent, "%")
}
```

Variabel `percent` nilainya didapat dari hasil perhitungan, dan hanya bisa digunakan di deretan blok seleksi kondisi itu saja.

Deklarasi variabel temporary hanya bisa dilakukan lewat metode type inference yang menggunakan tanda `:=`. Penggunaan keyword `var` disitu tidak diperbolehkan karena akan menyebabkan error.

### A.12.3. Seleksi Kondisi Menggunakan Keyword `switch - case`

Switch merupakan seleksi kondisi yang sifatnya fokus pada satu variabel, lalu kemudian di-cek nilainya. Contoh sederhananya seperti penentuan apakah nilai variabel `x` adalah: `1`, `2`, `3`, atau lainnya.

```
var point = 6

switch point {
case 8:
    fmt.Println("perfect")
case 7:
    fmt.Println("awesome")
default:
    fmt.Println("not bad")
}
```

Pada kode di atas, tidak ada kondisi atau `case` yang terpenuhi karena nilai variabel `point` tetap `6`. Ketika hal seperti ini terjadi, blok kondisi `default` dipanggil. Bisa dibilang bahwa `default` merupakan `else` dalam sebuah `switch`.

Perlu diketahui, `switch` pada pemrograman Go memiliki perbedaan dibanding bahasa lain. Di Go, ketika sebuah `case` terpenuhi, tidak akan dilanjutkan ke pengecekan `case` selanjutnya, meskipun tidak ada keyword `break` di situ. Konsep ini berkebalikan dengan `switch` pada umumnya, yang ketika sebuah `case` terpenuhi, maka akan tetap dilanjut mengecek `case` selanjutnya kecuali ada keyword `break`.

### A.12.4. Pemanfaatan `case` Untuk Banyak Kondisi

Sebuah `case` dapat menampung banyak kondisi. Cara penerapannya yaitu dengan menuliskan nilai pembanding-pembanding variabel yang di-switch setelah keyword `case` dipisah tanda koma ( , ).

```
var point = 6

switch point {
case 8:
```

```

    fmt.Println("perfect")
case 7, 6, 5, 4:
    fmt.Println("awesome")
default:
    fmt.Println("not bad")
}

```

Kondisi `case 7, 6, 5, 4:` akan terpenuhi ketika nilai variabel `point` adalah 7 atau 6 atau 5 atau 4.

## A.12.5. Kurung Kurawal Pada Keyword `case` & `default`

Tanda kurung kurawal (`{ }`) bisa diterapkan pada keyword `case` dan `default`. Tanda ini opsional, boleh dipakai boleh tidak. Bagus jika dipakai pada blok kondisi yang didalamnya ada banyak statement, kode akan terlihat lebih rapi dan mudah di-maintain.

Perhatikan kode berikut, bisa dilihat pada keyword `default` terdapat kurung kurawal yang mengapit 2 statement didalamnya.

```

var point = 6

switch point {
case 8:
    fmt.Println("perfect")
case 7, 6, 5, 4:
    fmt.Println("awesome")
default:
{
    fmt.Println("not bad")
    fmt.Println("you can be better!")
}
}

```

## A.12.6. Switch Dengan Gaya `if` - `else`

Uniknya di Golang, switch bisa digunakan dengan gaya ala if-else. Nilai yang akan dibandingkan tidak dituliskan setelah keyword `switch`, melainkan akan ditulis langsung dalam bentuk perbandingan dalam keyword `case`.

Pada kode di bawah ini, kode program switch di atas diubah ke dalam gaya `if-else`. Variabel `point` dihilangkan dari keyword `switch`, lalu kondisi-kondisinya dituliskan di tiap `case`.

```

var point = 6

switch {
case point == 8:
    fmt.Println("perfect")
case (point < 8) && (point > 3):
    fmt.Println("awesome")
default:
{
    fmt.Println("not bad")
    fmt.Println("you need to learn more")
}
}

```

## A.12.7. Penggunaan Keyword `fallthrough` Dalam `switch`

Seperti yang sudah dijelaskan sebelumnya, bahwa switch pada Golang memiliki perbedaan dengan bahasa lain. Ketika sebuah `case` terpenuhi, pengecekan kondisi tidak akan diteruskan ke `case`-`case` selanjutnya.

Keyword `fallthrough` digunakan untuk memaksa proses pengecekan diteruskan ke `case` selanjutnya.

```
var point = 6

switch {
case point == 8:
    fmt.Println("perfect")
case (point < 8) && (point > 3):
    fmt.Println("awesome")
    fallthrough
case point < 5:
    fmt.Println("you need to learn more")
default:
{
    fmt.Println("not bad")
    fmt.Println("you need to learn more")
}
}
```

Setelah pengecekan `case (point < 8) && (point > 3)` selesai, akan dilanjut ke pengecekan `case point < 5`, karena ada `fallthrough` di situ.

```
[novalagung:belajar-golang $ go run bab12.go
awesome
you need to learn more
novalagung:belajar-golang $ ]
```

## A.12.8. Seleksi Kondisi Bersarang

Seleksi kondisi bersarang adalah seleksi kondisi, yang berada dalam seleksi kondisi, yang mungkin juga berada dalam seleksi kondisi, dan seterusnya. Seleksi kondisi bersarang bisa dilakukan pada `if - else`, `switch`, ataupun kombinasi keduanya.

```
var point = 10

if point > 7 {
    switch point {
    case 10:
        fmt.Println("perfect!")
    default:
        fmt.Println("nice!")
    }
} else {
    if point == 5 {
        fmt.Println("not bad")
    } else if point == 3 {
        fmt.Println("keep trying")
    } else {
        fmt.Println("you can do it")
        if point == 0 {
            fmt.Println("try harder!")
        }
    }
}
```



## A.13. Perulangan

Perulangan adalah proses mengulang-ulang eksekusi blok kode tanpa henti, selama kondisi yang dijadikan acuan terpenuhi. Biasanya disiapkan variabel untuk iterasi atau variabel penanda kapan perulangan akan diberhentikan.

Di GoLang keyword perulangan hanya `for` saja, tetapi meski demikian, kemampuannya merupakan gabungan `for`, `foreach`, dan `while` ibarat bahasa pemrograman lain.

### A.13.1. Perulangan Menggunakan Keyword `for`

Ada beberapa cara standar menggunakan `for`. Cara pertama dengan memasukkan variabel counter perulangan beserta kondisinya setelah keyword. Perhatikan dan praktikan kode berikut.

```
for i := 0; i < 5; i++ {
    fmt.Println("Angka", i)
}
```

Perulangan di atas hanya akan berjalan ketika variabel `i` bernilai dibawah `5`, dengan ketentuan setiap kali perulangan, nilai variabel `i` akan di-iterasi atau ditambahkan 1 (`i++` artinya ditambah satu, sama seperti `i = i + 1`). Karena `i` pada awalnya bernilai 0, maka perulangan akan berlangsung 5 kali, yaitu ketika `i` bernilai 0, 1, 2, 3, dan 4.

```
[novalagung:belajar-golang $ go run bab13.go
Angka 0
Angka 1
Angka 2
Angka 3
Angka 4
novalagung:belajar-golang $ ]
```

### A.13.2. Penggunaan Keyword `for` Dengan Argumen Hanya Kondisi

Cara ke-2 adalah dengan menuliskan kondisi setelah keyword `for` (hanya kondisi). Deklarasi dan iterasi variabel counter tidak dituliskan setelah keyword, hanya kondisi perulangan saja. Konsepnya mirip seperti `while` milik bahasa pemrograman lain.

Kode berikut adalah contoh `for` dengan argumen hanya kondisi (seperti `if`), output yang dihasilkan sama seperti penerapan `for` cara pertama.

```
var i = 0

for i < 5 {
    fmt.Println("Angka", i)
    i++
}
```

### A.13.3. Penggunaan Keyword `for` Tanpa Argumen

Cara ke-3 adalah `for` ditulis tanpa kondisi. Dengan ini akan dihasilkan perulangan tanpa henti (sama dengan `for true`). Pemberhentian perulangan dilakukan dengan menggunakan keyword `break`.

```

var i = 0

for {
    fmt.Println("Angka", i)

    i++
    if i == 5 {
        break
    }
}

```

Dalam perulangan tanpa henti di atas, variabel `i` yang nilai awalnya `0` di-inkrementasi. Ketika nilai `i` sudah mencapai `5`, keyword `break` digunakan, dan perulangan akan berhenti.

## A.13.4. Penggunaan Keyword `for` - `range`

Cara ke-4 adalah perulangan dengan menggunakan kombinasi keyword `for` dan `range`. Cara ini biasa digunakan untuk me-looping data bertipe array. Detailnya akan dibahas dalam bab selanjutnya (bab 14).

## A.13.5. Penggunaan Keyword `break` & `continue`

Keyword `break` digunakan untuk menghentikan secara paksa sebuah perulangan, sedangkan `continue` dipakai untuk memaksa maju ke perulangan berikutnya.

Berikut merupakan contoh penerapan `continue` dan `break`. Kedua keyword tersebut dimanfaatkan untuk menampilkan angka genap berurutan yang lebih besar dari 0 dan dibawah 8.

```

for i := 1; i <= 10; i++ {
    if i % 2 == 1 {
        continue
    }

    if i > 8 {
        break
    }

    fmt.Println("Angka", i)
}

```

Kode di atas akan lebih mudah dicerna jika dijelaskan secara berurutan. Berikut adalah penjelasannya.

1. Dilakukan perulangan mulai angka 1 hingga 10 dengan `i` sebagai variabel iterasi.
2. Ketika `i` adalah ganjil (dapat diketahui dari `i % 2`, jika hasilnya `1`, berarti ganjil), maka akan dipaksa lanjut ke perulangan berikutnya.
3. Ketika `i` lebih besar dari 8, maka perulangan akan berhenti.
4. Nilai `m` ditampilkan.

```

[novalagung:belajar-golang $ go run bab13.go
Angka 2
Angka 4
Angka 6
Angka 8
novalagung:belajar-golang $ ]

```

## A.13.6. Perulangan Bersarang

Tak hanya seleksi kondisi yang bisa bersarang, perulangan juga bisa. Cara pengaplikasiannya kurang lebih sama, tinggal tulis blok statement perulangan didalam perulangan.

```
for i := 0; i < 5; i++ {
    for j := i; j < 5; j++ {
        fmt.Println(j, " ")
    }

    fmt.Println()
}
```

Pada kode di atas, untuk pertama kalinya fungsi `fmt.Println()` dipanggil tanpa disisipkan parameter. Cara seperti ini bisa digunakan untuk menampilkan baris baru. Kegunaannya sama seperti output dari statement `fmt.Println("\n")`.

```
[novalagung:belajar-golang $ go run bab13.go
0 1 2 3 4
1 2 3 4
2 3 4
3 4
4
novalagung:belajar-golang $ ]
```

## A.13.7. Pemanfaatan Label Dalam Perulangan

Di perulangan bersarang, `break` dan `continue` akan berlaku pada blok perulangan dimana ia digunakan saja. Ada cara agar kedua keyword ini bisa tertuju pada perulangan terluar atau perulangan tertentu, yaitu dengan memanfaatkan teknik pemberian **label**.

Program untuk memunculkan matriks berikut merupakan contoh penerapan label perulangan.

```
outerLoop:
for i := 0; i < 5; i++ {
    for j := 0; j < 5; j++ {
        if i == 3 {
            break outerLoop
        }
        fmt.Println("matriks [", i, "][", j, "]", "\n")
    }
}
```

Tepat sebelum keyword `for` terluar, terdapat baris kode `outerLoop:`. Maksud dari kode tersebut adalah disiapkan sebuah label bernama `outerLoop` untuk `for` dibawahnya. Nama label bisa diganti dengan nama lain (dan harus diakhiri dengan tanda titik dua atau `colon ( : )`).

Pada `for` bagian dalam, terdapat seleksi kondisi untuk pengecekan nilai `i`. Ketika nilai tersebut sama dengan `3`, maka `break` dipanggil dengan target adalah perulangan yang dilabeli `outerLoop`, perulangan tersebut akan dihentikan.

```
[novalagung:belajar-golang $ go run bab13.go
matriks [0] [0]
matriks [0] [1]
matriks [0] [2]
matriks [0] [3]
matriks [0] [4]
matriks [1] [0]
matriks [1] [1]
matriks [1] [2]
matriks [1] [3]
matriks [1] [4]
matriks [2] [0]
matriks [2] [1]
matriks [2] [2]
matriks [2] [3]
matriks [2] [4]
novalagung:belajar-golang $ ]
```

## A.14. Array

Array adalah kumpulan data bertipe sama, yang disimpan dalam sebuah variabel. Array memiliki kapasitas yang nilainya ditentukan pada saat pembuatan, menjadikan elemen/data yang disimpan di array tersebut jumlahnya tidak boleh melebihi yang sudah dialokasikan. Default nilai tiap elemen array pada awalnya tergantung dari tipe datanya. Jika `int` maka default nya `0`, jika `bool` maka default-nya `false`, dan tipe data lain. Setiap elemen array memiliki indeks berupa angka yang merepresentasikan posisi urutan elemen tersebut. Indeks array dimulai dari 0.

Contoh penerapan array:

```
var names [4]string
names[0] = "trafalgar"
names[1] = "d"
names[2] = "water"
names[3] = "law"

fmt.Println(names[0], names[1], names[2], names[3])
```

Variabel `names` dideklarasikan sebagai `array string` dengan alokasi elemen 4 slot. Cara mengisi slot elemen array bisa dilihat di kode di atas, yaitu dengan langsung mengakses elemen menggunakan indeks, lalu mengisinya.

```
[novalagung:belajar-golang $ go run bab14.go
trafalgar d water law
novalagung:belajar-golang $ ]
```

### A.14.1. Pengisian Elemen Array yang Melebihi Alokasi Awal

Pengisian elemen array pada indeks yang tidak sesuai dengan alokasi menghasilkan error. Contoh sederhana, jika array memiliki 4 slot, maka pengisian nilai slot 5 seterusnya adalah tidak valid.

```
var names [4]string
names[0] = "trafalgar"
names[1] = "d"
names[2] = "water"
names[3] = "law"
names[4] = "ez" // baris kode ini menghasilkan error
```

Solusi dari masalah di atas adalah dengan menggunakan keyword `append`, yang di bab selanjutnya akan kita bahas.

### A.14.2. Inisialisasi Nilai Awal Array

Pengisian elemen array bisa dilakukan pada saat deklarasi variabel. Caranya dengan menuliskan data elemen dalam kurung kurawal setelah tipe data, dengan pembatas antar elemen adalah tanda koma ( , ).

```
var fruits = [4]string{"apple", "grape", "banana", "melon"}

fmt.Println("Jumlah element \t\t", len(fruits))
fmt.Println("Isi semua element \t", fruits)
```

Penggunaan fungsi `fmt.Println()` pada data array tanpa mengakses indeks tertentu, akan menghasilkan output dalam bentuk string dari semua array yang ada. Teknik ini biasa digunakan untuk **debugging** data array.

```
[novalagung:belajar-golang $ go run bab14.go
Jumlah element      4
Isi semua element  [apple grape banana melon]
novalagung:belajar-golang $ ]
```

Fungsi `len()` dipakai untuk menghitung jumlah elemen sebuah array.

### A.14.3. Inisialisasi Nilai Array Dengan Gaya Vertikal

Elemen array bisa dituliskan dalam bentuk horizontal (seperti yang sudah dicontohkan di atas) ataupun dalam bentuk vertikal.

```
var fruits [4]string

// cara horizontal
fruits = [4]string{"apple", "grape", "banana", "melon"}

// cara vertikal
fruits = [4]string{
    "apple",
    "grape",
    "banana",
    "melon",
}
```

Khusus untuk deklarasi array dengan cara vertikal, tanda koma wajib dituliskan setelah elemen, termasuk elemen terakhir. Jika tidak, maka akan muncul error.

### A.14.4. Inisialisasi Nilai Awal Array Tanpa Jumlah Elemen

Deklarasi array yang nilainya diset di awal, boleh tidak dituliskan jumlah lebar array-nya, cukup ganti dengan tanda 3 titik (`...`). Jumlah elemen akan dikalkulasi secara otomatis menyesuaikan data elemen yang diisikan.

```
var numbers = [...]int{2, 3, 2, 4, 3}

fmt.Println("data array \t:", numbers)
fmt.Println("jumlah elemen \t:", len(numbers))
```

Variabel `numbers` akan secara otomatis memiliki jumlah elemen `5`, karena pada saat deklarasi disiapkan 5 buah elemen.

```
[novalagung:belajar-golang $ go run bab14.go
data array      : [2 3 2 4 3]
jumlah elemen  : 5
novalagung:belajar-golang $ ]
```

### A.14.5. Array Multidimensi

Array multidimensi adalah array yang tiap elemennya juga berupa array (dan bisa seterusnya, tergantung kedalamannya dimensinya).

Cara deklarasi array multidimensi secara umum sama dengan cara deklarasi array biasa, dengan cara menuliskan data array dimensi selanjutnya sebagai elemen array dimensi sebelumnya.

Khusus untuk array yang merupakan sub dimensi atau elemen, boleh tidak dituliskan jumlah datanya. Contohnya bisa dilihat pada deklarasi variabel `numbers2` di kode berikut.

```

var numbers1 = [2][3]int{[3]int{3, 2, 3}, [3]int{3, 4, 5}}
var numbers2 = [2][3]int{{3, 2, 3}, {3, 4, 5}]

fmt.Println("numbers1", numbers1)
fmt.Println("numbers2", numbers2)

```

Kedua array di atas memiliki elemen yang sama.

```
[novalagung:belajar-golang $ go run bab14.go
numbers1 [[3 2 3] [3 4 5]]
numbers2 [[3 2 3] [3 4 5]]
novalagung:belajar-golang $ ]
```

## A.14.6. Perulangan Elemen Array Menggunakan Keyword for

Keyword `for` dan array memiliki hubungan yang sangat erat. Dengan memanfaatkan perulangan menggunakan keyword ini, elemen-elemen dalam array bisa didapat.

Ada beberapa cara yang bisa digunakan untuk me-looping data array, yg pertama adalah dengan memanfaatkan variabel iterasi perulangan untuk mengakses elemen berdasarkan indeks-nya. Contoh:

```

var fruits = [4]string{"apple", "grape", "banana", "melon"}

for i := 0; i < len(fruits); i++ {
    fmt.Printf("elemen %d : %s\n", i, fruits[i])
}

```

Perulangan di atas dijalankan sebanyak jumlah elemen array `fruits` (bisa diketahui dari kondisi `i < len(fruits)`). Di tiap perulangan, elemen array diakses lewat variabel iterasi `i`.

```
[novalagung:belajar-golang $ go run bab14.go
elemen 0 : apple
elemen 1 : grape
elemen 2 : banana
elemen 3 : melon
novalagung:belajar-golang $ ]
```

## A.14.7. Perulangan Elemen Array Menggunakan Keyword for - range

Ada cara yang lebih sederhana me-looping data array, dengan menggunakan keyword `for - range`. Contoh pengaplikasiannya bisa dilihat di kode berikut.

```

var fruits = [4]string{"apple", "grape", "banana", "melon"}

for i, fruit := range fruits {
    fmt.Printf("elemen %d : %s\n", i, fruit)
}

```

Array `fruits` diambil elemen-nya secara berurutan. Nilai tiap elemen ditampung variabel oleh `fruit` (tanpa huruf `s`), sedangkan indeks nya ditampung variabel `i`.

Output program di atas, sama dengan output program sebelumnya, hanya cara yang digunakan berbeda.

## A.14.8. Penggunaan Variabel Underscore `_` Dalam `for - range`

Kadang kala ketika *looping* menggunakan `for - range`, ada kemungkinan dimana data yang dibutuhkan adalah elemen-nya saja, indeks-nya tidak. Sedangkan kode di atas, `range` mengembalikan 2 data, yaitu indeks dan elemen.

Seperti yang sudah diketahui, bahwa di Golang tidak memperbolehkan adanya variabel yang menaggur atau tidak dipakai. Jika dipaksakan, error akan muncul, contohnya seperti kode berikut.

```
var fruits = [4]string{"apple", "grape", "banana", "melon"}

for i, fruit := range fruits {
    fmt.Printf("nama buah : %s\n", fruit)
}
```

Hasil dari kode program di atas:

```
[novalagung:belajar-golang $ go run bab14.go
# command-line-arguments
./bab14.go:8: i declared and not used
novalagung:belajar-golang $ ]
```

Disinilah salah satu kegunaan variabel pengangguran, atau underscore (`_`). Tampung saja nilai yang tidak ingin digunakan ke underscore.

```
var fruits = [4]string{"apple", "grape", "banana", "melon"}

for _, fruit := range fruits {
    fmt.Printf("nama buah : %s\n", fruit)
}
```

Pada kode di atas, yang sebelumnya adalah variabel `i` diganti dengan `_`, karena kebetulan variabel `i` tidak digunakan.

```
[novalagung:belajar-golang $ go run bab14.go
nama buah : apple
nama buah : grape
nama buah : banana
nama buah : melon
novalagung:belajar-golang $ ]
```

Jika yang dibutuhkan hanya indeks elemen-nya saja, bisa gunakan 1 buah variabel setelah keyword `for`.

```
for i, _ := range fruits { }
// atau
for i := range fruits { }
```

## A.14.9. Alokasi Elemen Array Menggunakan Keyword `make`

Deklarasi sekaligus alokasi data array juga bisa dilakukan lewat keyword `make`.

```
var fruits = make([]string, 2)
fruits[0] = "apple"
fruits[1] = "mango"

fmt.Println(fruits) // [apple mango]
```

Parameter pertama keyword `make` diisi dengan tipe data elemen array yang diinginkan, parameter kedua adalah jumlah elemennya. Pada kode di atas, variabel `fruits` tercetak sebagai array string dengan alokasi 2 slot.

## A.15. Slice

**Slice** adalah *reference* elemen array. Slice bisa dibuat, atau bisa juga dihasilkan dari manipulasi sebuah array ataupun slice lainnya. Karena merupakan data *reference*, menjadikan perubahan data di tiap elemen slice akan berdampak pada slice lain yang memiliki alamat memori yang sama.

### A.15.1. Inisialisasi Slice

Cara pembuatan slice mirip seperti pembuatan array, bedanya tidak perlu mendefinisikan jumlah elemen ketika awal deklarasi. Pengaksesan nilai elemen-nya juga sama. Kode berikut adalah contoh pembuatan slice.

```
var fruits = []string{"apple", "grape", "banana", "melon"}
fmt.Println(fruits[0]) // "apple"
```

Salah satu perbedaan slice dan array bisa diketahui pada saat deklarasi variabel-nya, jika jumlah elemen tidak dituliskan, maka variabel tersebut adalah slice.

```
var fruitsA = []string{"apple", "grape"}      // slice
var fruitsB = [2]string{"banana", "melon"}     // array
var fruitsC = [...]string{"papaya", "grape"}   // array
```

### A.15.2. Hubungan Slice Dengan Array & Operasi Slice

Kalau perbedannya hanya di penentuan alokasi pada saat inisialisasi, kenapa tidak menggunakan satu istilah saja? atau adakah perbedaan lainnya?

Sebenarnya slice dan array tidak bisa dibedakan karena merupakan sebuah kesatuan. Array adalah kumpulan nilai atau elemen, sedang slice adalah referensi tiap elemen tersebut.

Slice bisa dibentuk dari array yang sudah didefinisikan, caranya dengan memanfaatkan teknik **2 index** untuk mengambil elemen-nya. Contoh bisa dilihat pada kode berikut.

```
var fruits = []string{"apple", "grape", "banana", "melon"}
var newFruits = fruits[0:2]

fmt.Println(newFruits) // ["apple", "grape"]
```

Kode `fruits[0:2]` maksudnya adalah pengaksesan elemen dalam slice `fruits` yang **dimulai dari indeks ke-0, hingga elemen sebelum indeks ke-2**. Elemen yang memenuhi kriteria tersebut akan didapat, untuk kemudian disimpan pada variabel lain sebagai slice baru. Pada contoh di atas, `newFruits` adalah slice baru yang tercetak dari slice `fruits`, dengan isi 2 elemen, yaitu `"apple"` dan `"grape"`.

```
[novalagung:belajar-golang $ go run bab15.go
 [apple grape]
 novalagung:belajar-golang $ ]
```

Ketika mengakses elemen array menggunakan satu buah indeks (seperti `data[2]`), nilai yang didapat merupakan hasil **copy** dari referensi aslinya. Berbeda dengan pengaksesan elemen menggunakan 2 indeks (seperti `data[0:2]`), nilai yang didapat adalah *reference* elemen atau slice.

Tidak apa jikalau pembaca masih bingung, di bawah akan dijelaskan lebih mendetail lagi tentang slice dan *reference*

Tabel berikut adalah list operasi operasi menggunakan teknik 2 indeks yang bisa dilakukan.

```
var fruits = []string{"apple", "grape", "banana", "melon"}
```

Kode	Output	Penjelasan
fruits[0:2]	[apple, grape]	semua elemen mulai indeks ke-0, hingga sebelum indeks ke-2
fruits[0:4]	[apple, grape, banana, melon]	semua elemen mulai indeks ke-0, hingga sebelum indeks ke-4
fruits[0:0]	[]	menghasilkan slice kosong, karena tidak ada elemen sebelum indeks ke-0
fruits[4:4]	[]	menghasilkan slice kosong, karena tidak ada elemen yang dimulai dari indeks ke-4
fruits[4:0]	[]	error, pada penulisan <code>fruits[a,b]</code> nilai <code>a</code> harus lebih besar atau sama dengan <code>b</code>
fruits[:]	[apple, grape, banana, melon]	semua elemen
fruits[2:]	[banana, melon]	semua elemen mulai indeks ke-2
fruits[:-2]	[apple, grape]	semua elemen hingga sebelum indeks ke-2

### A.15.3. Slice Merupakan Tipe Data Reference

Slice merupakan tipe data *reference* atau referensi. Artinya jika ada slice baru yang terbentuk dari slice lama, maka data elemen slice yang baru akan memiliki alamat memori yang sama dengan elemen slice lama. Setiap perubahan yang terjadi di elemen slice baru, akan berdampak juga pada elemen slice lama yang memiliki referensi yang sama.

Program berikut merupakan pembuktian tentang teori yang baru kita bahas. Kita akan mencoba mengubah data elemen slice baru, yang terbentuk dari slice lama.

```
var fruits = []string{"apple", "grape", "banana", "melon"}

var aFruits = fruits[0:3]
var bFruits = fruits[1:4]

var aaFruits = aFruits[1:2]
var baFruits = bFruits[0:1]

fmt.Println(fruits)    // [apple grape banana melon]
fmt.Println(aFruits)  // [apple grape banana]
fmt.Println(bFruits)  // [grape banana melon]
fmt.Println(aaFruits) // [grape]
fmt.Println(baFruits) // [grape]

// Buah "grape" diubah menjadi "pinnacle"
baFruits[0] = "pinnacle"

fmt.Println(fruits)    // [apple pineapple banana melon]
fmt.Println(aFruits)  // [apple pineapple banana]
fmt.Println(bFruits)  // [pineapple banana melon]
fmt.Println(aaFruits) // [pineapple]
fmt.Println(baFruits) // [pineapple]
```

Sekilas bisa kita lihat bahwa setelah slice yang isi datanya adalah `grape` di-ubah menjadi `pinnacle`, semua slice pada 4 variabel lainnya juga ikut berubah.

Variabel `aFruits`, `bFruits` merupakan slice baru yang terbentuk dari variabel `fruits`. Dengan menggunakan dua slice baru tersebut, diciptakan lagi slice lainnya, yaitu `aaFruits`, dan `baFruits`. Kelima slice tersebut ditampilkan nilainya.

Selanjutnya, nilai dari `baFruits[0]` diubah, dan 5 slice tadi ditampilkan lagi. Hasilnya akan ada banyak slice yang elemennya ikut berubah. Yaitu elemen-elemen yang referensi-nya sama dengan referensi elemen `baFruits[0]`.

```
[novalagung:belajar-golang $ go run bab15.go
fruits          [apple grape banana melon]
aFruits         [apple grape banana]
bFruits         [grape banana melon]
aaFruits        [grape]
baFruits        [grape]

Buah "grape" diubah menjadi "pinnapple"

fruits          [apple pinnapple banana melon]
aFruits         [apple pinnapple banana]
bFruits         [pinnapple banana melon]
aaFruits        [pinnapple]
baFruits        [pinnapple]
novalagung:belajar-golang $ ]
```

Bisa dilihat pada output di atas, elemen yang sebelumnya bernilai "grape" pada variabel `fruits`, `aFruits`, `bFruits`, `aaFruits`, dan `baFruits`; kesemuanya berubah menjadi "pinnapple", karena memiliki referensi yang sama.

Pembahasan mengenai dasar slice sepertinya sudah cukup, selanjutnya kita akan membahas tentang beberapa *built in function* bawaan Golang, yang bisa dimanfaatkan untuk keperluan operasi slice.

## A.15.4. Fungsi `len()`

Fungsi `len()` digunakan untuk menghitung jumlah elemen slice yang ada. Sebagai contoh jika sebuah variabel adalah slice dengan data 4 buah, maka fungsi ini pada variabel tersebut akan mengembalikan angka 4.

```
var fruits = []string{"apple", "grape", "banana", "melon"}
fmt.Println(len(fruits)) // 4
```

## A.15.5. Fungsi `cap()`

Fungsi `cap()` digunakan untuk menghitung lebar atau kapasitas maksimum slice. Nilai kembalian fungsi ini untuk slice yang baru dibuat pasti sama dengan `len`, tapi bisa berubah seiring operasi slice yang dilakukan. Agar lebih jelas, silakan disimak kode berikut.

```
var fruits = []string{"apple", "grape", "banana", "melon"}
fmt.Println(len(fruits)) // len: 4
fmt.Println(cap(fruits)) // cap: 4

var aFruits = fruits[0:3]
fmt.Println(len(aFruits)) // len: 3
fmt.Println(cap(aFruits)) // cap: 4

var bFruits = fruits[1:4]
fmt.Println(len(bFruits)) // len: 3
fmt.Println(cap(bFruits)) // cap: 3
```

Variabel `fruits` disiapkan di awal dengan jumlah elemen 4, fungsi `len(fruits)` dan `cap(fruits)` pasti hasilnya 4.

Variabel `aFruits` dan `bFruits` merupakan slice baru berisikan 3 buah elemen milik slice `fruits`. Variabel `aFruits` mengambil elemen index 0, 1, 2; sedangkan `bFruits` 1, 2, 3.

Fungsi `len()` menghasilkan angka 3, karena jumlah elemen kedua slice ini adalah 3. Tetapi `cap(aFruits)` menghasilkan angka yang berbeda, yaitu 4 untuk `aFruits` dan 3 untuk `bFruits`. kenapa? jawabannya bisa dilihat pada tabel berikut.

Kode	Output	<code>len()</code>	<code>cap()</code>
<code>fruits[0:4]</code>	[ buah buah buah buah ]	4	4
<code>aFruits[0:3]</code>	[ buah buah buah ---- ]	3	4
<code>bFruits[1:3]</code>	---- [ buah buah buah ]	3	3

Kita analogikan slicing 2 index menggunakan **x** dan **y**.

```
fruits[x:y]
```

**Slicing** yang dimulai dari indeks **0** hingga **y** akan mengembalikan elemen-elemen mulai indeks **0** hingga sebelum indeks **y**, dengan lebar kapasitas adalah sama dengan slice aslinya.

Sedangkan slicing yang dimulai dari indeks **x**, yang dimana nilai **x** adalah lebih dari **0**, membuat elemen ke-**x** slice yang diambil menjadi elemen ke-0 slice baru. Hal inilah yang membuat kapasitas slice berubah.

## A.15.6. Fungsi `append()`

Fungsi `append()` digunakan untuk menambahkan elemen pada slice. Elemen baru tersebut diposisikan setelah indeks paling akhir. Nilai balik fungsi ini adalah slice yang sudah ditambahkan nilai barunya. Contoh penggunaannya bisa dilihat di kode berikut.

```
var fruits = []string{"apple", "grape", "banana"}
var cFruits = append(fruits, "papaya")

fmt.Println(fruits) // ["apple", "grape", "banana"]
fmt.Println(cFruits) // ["apple", "grape", "banana", "papaya"]
```

Ada 3 hal yang perlu diketahui dalam penggunaan fungsi ini.

- Ketika jumlah elemen dan lebar kapasitas adalah sama (`len(fruits) == cap(fruits)`), maka elemen baru hasil `append()` merupakan referensi baru.
- Ketika jumlah elemen lebih kecil dibanding kapasitas (`len(fruits) < cap(fruits)`), elemen baru tersebut ditempatkan kedalam cakupan kapasitas, menjadikan semua elemen slice lain yang referensi-nya sama akan berubah nilainya.

Agar lebih jelas silakan perhatikan contoh berikut.

```
var fruits = []string{"apple", "grape", "banana"}
var bFruits = fruits[0:2]

fmt.Println(cap(bFruits)) // 3
fmt.Println(len(bFruits)) // 2

fmt.Println(fruits) // ["apple", "grape", "banana"]
fmt.Println(bFruits) // ["apple", "grape"]

var cFruits = append(bFruits, "papaya")
```

```
fmt.Println(fruits) // ["apple", "grape", "papaya"]
fmt.Println(bFruits) // ["apple", "grape"]
fmt.Println(cFruits) // ["apple", "grape", "papaya"]
```

Pada contoh di atas bisa dilihat, elemen indeks ke-2 slice `fruits` nilainya berubah setelah ada penggunaan keyword `append()` pada `bFruits`. Slice `bFruits` kapasitasnya adalah **3** sedang jumlah datanya hanya **2**. Karena `len(bFruits) < cap(bFruits)`, maka elemen baru yang dihasilkan, terdeteksi sebagai perubahan nilai pada referensi yang lama (referensi elemen indeks ke-2 slice `fruits`), membuat elemen yang referensinya sama, nilainya berubah.

## A.15.7. Fungsi `copy()`

Fungsi `copy()` digunakan untuk men-copy elements slice pada `src` (parameter ke-2), ke `dst` (parameter pertama).

```
copy(dst, src)
```

Jumlah element yang di-copy dari `src` adalah sejumlah lebar slice `dst` (atau `len(dst)`). Jika jumlah slice pada `src` lebih kecil dari `dst`, maka akan ter-copy semua. Lebih jelasnya silakan perhatikan contoh berikut.

```
dst := make([]string, 3)
src := []string{"watermelon", "pinnapple", "apple", "orange"}
n := copy(dst, src)

fmt.Println(dst) // watermelon pinnapple apple
fmt.Println(src) // watermelon pinnapple apple orange
fmt.Println(n) // 3
```

Pada kode di atas variabel slice `dst` dipersiapkan dengan lebar adalah 3 elements. Slice `src` yang isinya 4 elements, di-copy ke `dst`. Menjadikan isi slice `dst` sekarang adalah 3 buah elements yang sama dengan 3 buah elements `src`, hasil dari operasi `copy()`.

Yang ter-copy hanya 3 buah (meski `src` memiliki 4 elements) hal ini karena `copy()` hanya meng-copy elements sebanyak `len(dst)`.

Fungsi `copy()` mengembalikan informasi angka, representasi dari jumlah element yang berhasil di-copy.

Pada contoh kedua berikut, `dst` merupakan slice yang sudah ada isinya, 3 buah elements. Variabel `src` yang juga merupakan slice dengan isi dua elements, di-copy ke `dst`. Karena operasi `copy()` akan meng-copy sejumlah `len(dst)`, maka semua elements `src` akan ter-copy **karena jumlahnya dibawah atau sama dengan lebar dst**.

```
dst := []string{"potato", "potato", "potato"}
src := []string{"watermelon", "pinnapple"}
n := copy(dst, src)

fmt.Println(dst) // watermelon pinnapple potato
fmt.Println(src) // watermelon pinnapple
fmt.Println(n) // 2
```

Jika dilihat pada kode di atas, isi `dst` masih tetap 3 elements, tapi dua elements pertama adalah sama dengan `src`. Element terakhir `dst` isinya tidak berubah, tetapi `potato`, hal ini karena proses copy hanya memutasi element ke-1 dan ke-2 milik `dst`, karena memang pada `src` hanya dua itu elements-nya.

## A.15.8. Pengaksesan Elemen Slice Dengan 3 Indeks

**3 index** adalah teknik slicing elemen yang sekaligus menentukan kapasitasnya. Cara menggunakannya yaitu dengan menyiapkan angka kapasitas di belakang, seperti `fruits[0:1:1]`. Angka kapasitas yang diisikan tidak boleh melebihi kapasitas slice yang akan di slicing.

Berikut merupakan contoh penerapannya.

```
var fruits = []string{"apple", "grape", "banana"}  
var aFruits = fruits[0:2]  
var bFruits = fruits[0:2:2]  
  
fmt.Println(fruits)      // ["apple", "grape", "banana"]  
fmt.Println(len(fruits)) // len: 3  
fmt.Println(cap(fruits)) // cap: 3  
  
fmt.Println(aFruits)      // ["apple", "grape"]  
fmt.Println(len(aFruits)) // len: 2  
fmt.Println(cap(aFruits)) // cap: 3  
  
fmt.Println(bFruits)      // ["apple", "grape"]  
fmt.Println(len(bFruits)) // len: 2  
fmt.Println(cap(bFruits)) // cap: 2
```

## A.16. Map

**Map** adalah tipe data asosiatif yang ada di Golang, berbentuk *key-value*. Untuk setiap data (atau value) yang disimpan, disiapkan juga key-nya. Key harus unik, karena digunakan sebagai penanda (atau identifier) untuk pengaksesan value yang bersangkutan.

Kalau dilihat, `map` mirip seperti slice, hanya saja indeks yang digunakan untuk pengaksesan bisa ditentukan sendiri tipe-nya (indeks tersebut adalah key).

### A.16.1. Penggunaan Map

Cara menggunakan map cukup dengan menuliskan keyword `map` diikuti tipe data key dan value-nya. Agar lebih mudah dipahami, silakan perhatikan contoh di bawah ini.

```
var chicken map[string]int
chicken = map[string]int{}

chicken["januari"] = 50
chicken["februari"] = 40

fmt.Println("januari", chicken["januari"]) // januari 50
fmt.Println("mei",     chicken["mei"])      // mei 0
```

Variabel `chicken` dideklarasikan sebagai map, dengan tipe data key adalah `string` dan value-nya `int`. Dari kode tersebut bisa dilihat bagaimana cara penggunaan keyword `map`.

Kode `map[string]int` maknanya adalah, tipe data `map` dengan key bertipe `string` dan value bertipe `int`.

Default nilai variabel `map` adalah `nil`. Oleh karena itu perlu dilakukan inisialisasi nilai default di awal, caranya cukup dengan tambahkan kurung kurawal pada akhir tipe, contoh seperti pada kode di atas: `map[string]int{}`.

Cara menge-set nilai pada sebuah map adalah dengan menuliskan variabel-nya, kemudian disisipkan `key` pada kurung siku variabel (mirip seperti cara pengaksesan elemen slice), lalu isi nilainya. Contohnya seperti `chicken["februari"] = 40`. Sedangkan cara pengambilan value adalah cukup dengan menyisipkan `key` pada kurung siku variabel.

Pengisian data pada map bersifat **overwrite**, ketika variabel sudah memiliki item dengan key yang sama, maka value lama akan ditimpas dengan value baru.

```
[novalagung:belajar-golang $ go run bab16.go
januari 50
mei 0
novalagung:belajar-golang $ ]
```

Pada pengaksesan item menggunakan key yang belum tersimpan di map, akan dikembalikan nilai default tipe data value-nya. Contohnya seperti pada kode di atas, `chicken["mei"]` menghasilkan nilai 0 (nilai default tipe `int`), karena belum ada item yang tersimpan menggunakan key `"mei"`.

### A.16.2. Inisialisasi Nilai Map

Nilai variabel bertipe map bisa didefinisikan di awal, caranya dengan menambahkan kurung kurawal setelah tipe data, lalu menuliskan key dan value didalamnya. Cara ini sekilas mirip dengan definisi nilai array/slice namun dalam bentuk key-value.

```
// cara vertikal
var chicken1 = map[string]int{"januari": 50, "februari": 40}

// cara horizontal
var chicken2 = map[string]int{
    "januari": 50,
    "februari": 40,
}
```

Key dan value dituliskan dengan pembatas tanda titik dua ( : ). Sedangkan tiap itemnya dituliskan dengan pembatas tanda koma ( , ). Khusus deklarasi dengan gaya vertikal, tanda koma perlu dituliskan setelah item terakhir.

Variabel `map` bisa diinisialisasi dengan tanpa nilai awal, caranya menggunakan tanda kurung kurawal, contoh: `map[string]int{}`. Atau bisa juga dengan menggunakan keyword `make` dan `new`. Contohnya bisa dilihat pada kode berikut. Ketiga cara di bawah ini intinya adalah sama.

```
var chicken3 = map[string]int{}
var chicken4 = make(map[string]int)
var chicken5 = *new(map[string]int)
```

Khusus inisialisasi data menggunakan keyword `new`, yang dihasilkan adalah data pointer. Untuk mengambil nilai aslinya bisa dengan menggunakan tanda asterisk ( \* ). Topik pointer akan dibahas lebih detail ketika sudah masuk bab 22.

### A.16.3. Iterasi Item Map Menggunakan `for - range`

Item variabel `map` bisa di iterasi menggunakan `for - range`. Cara penerapannya masih sama seperti pada slice, pembedanya data yang dikembalikan di tiap perulangan adalah key dan value, bukan indeks dan elemen. Contohnya bisa dilihat pada kode berikut.

```
var chicken = map[string]int{
    "januari": 50,
    "februari": 40,
    "maret": 34,
    "april": 67,
}

for key, val := range chicken {
    fmt.Println(key, " \t:", val)
}
```

```
[novalagung:belajar-golang $ go run bab16.go
januari      : 50
februari     : 40
maret        : 34
april        : 67
novalagung:belajar-golang $ ]
```

### A.16.4. Menghapus Item Map

Fungsi `delete()` digunakan untuk menghapus item dengan key tertentu pada variabel map. Cara penggunaannya, dengan memasukan objek map dan key item yang ingin dihapus sebagai parameter.

```
var chicken = map[string]int{"januari": 50, "februari": 40}

fmt.Println(len(chicken)) // 2
fmt.Println(chicken)
```

```
delete(chicken, "januari")

fmt.Println(len(chicken)) // 1
fmt.Println(chicken)
```

Item yang memiliki key "januari" dalam variabel chicken akan dihapus.

```
[novalagung:belajar-golang $ go run bab16.go
4 items : map[april:67 januari:50 februari:40 maret:34]
3 items : map[februari:40 maret:34 april:67]
novalagung:belajar-golang $ ]
```

Fungsi `len()` jika digunakan pada map akan mengembalikan jumlah item.

## A.16.5. Deteksi Keberadaan Item Dengan Key Tertentu

Ada cara untuk mengetahui apakah dalam sebuah variabel map terdapat item dengan key tertentu atau tidak, yaitu dengan memanfaatkan 2 variabel sebagai penampung nilai kembalian pengaksesan item. Variabel ke-2 akan berisikan nilai `bool` yang menunjukkan ada atau tidaknya item yang dicari.

```
var chicken = map[string]int{"januari": 50, "februari": 40}
var value, isExist = chicken["mei"]

if isExist {
    fmt.Println(value)
} else {
    fmt.Println("item is not exists")
}
```

## A.16.6. Kombinasi Slice & Map

Slice dan `map` bisa dikombinasikan, dan sering digunakan pada banyak kasus, contohnya seperti data array yang berisikan informasi siswa, dan banyak lainnya.

Cara menggunakan cukup mudah, contohnya seperti `[]map[string]int`, artinya slice yang tipe tiap elemen-nya adalah `map[string]int`.

Agar lebih jelas, silakan praktikan contoh berikut.

```
var chickens = []map[string]string{
    map[string]string{"name": "chicken blue",   "gender": "male"},
    map[string]string{"name": "chicken red",    "gender": "male"},
    map[string]string{"name": "chicken yellow", "gender": "female"},
}

for _, chicken := range chickens {
    fmt.Println(chicken["gender"], chicken["name"])
}
```

Variabel `chickens` di atas berisikan informasi bertipe `map[string]string`, yang kebetulan tiap elemen memiliki 2 key yang sama.

Jika anda menggunakan versi go terbaru, cara deklarasi slice-map bisa dipersingkat, tipe tiap elemen tidak wajib untuk dituliskan.

```
var chickens = []map[string]string{
    {"name": "chicken blue",   "gender": "male"},
```

```
{"name": "chicken red", "gender": "male"},  
 {"name": "chicken yellow", "gender": "female"},  
 }
```

Dalam `[]map[string]string`, tiap elemen bisa saja memiliki key yang berbeda-beda, sebagai contoh seperti kode berikut.

```
var data = []map[string]string{  
    {"name": "chicken blue", "gender": "male", "color": "brown"},  
    {"address": "mangga street", "id": "k001"},  
    {"community": "chicken lovers"}  
}
```

## A.17. Fungsi

Fungi merupakan aspek penting dalam pemrograman. Definisi fungsi sendiri adalah sekumpulan blok kode yang dibungkus dengan nama tertentu. Penerapan fungsi yang tepat akan menjadikan kode lebih modular dan juga *dry* (kependekan dari *don't repeat yourself*), tak perlu menuliskan banyak kode yang kegunaannya berkali-kali, cukup sekali saja lalu panggil sesuai kebutuhan.

Di bab ini kita akan belajar tentang penggunaan fungsi di Golang.

### A.17.1. Penerapan Fungsi

Sebenarnya tanpa sadar, kita sudah menerapkan fungsi di bab-bab sebelum ini, yaitu pada fungsi `main`. Fungsi `main` merupakan fungsi yang paling utama pada program Golang.

Cara membuat fungsi cukup mudah, yaitu dengan menuliskan keyword `func`, diikuti setelahnya nama fungsi, kurung yang berisikan parameter, dan kurung kurawal untuk membungkus blok kode.

Parameter sendiri adalah variabel yang disisipkan pada saat pemanggilan fungsi.

Silakan lihat dan praktikan kode tentang implementasi fungsi berikut.

```
package main

import "fmt"
import "strings"

func main() {
    var names = []string{"John", "Wick"}
    printMessage("halo", names)
}

func printMessage(message string, arr []string) {
    var nameString = strings.Join(arr, " ")
    fmt.Println(message, nameString)
}
```

Pada kode di atas, sebuah fungsi baru dibuat dengan nama `printMessage` memiliki 2 buah parameter yaitu string `message` dan slice string `arr`.

Fungsi tersebut dipanggil dalam `main`, dengan disisipkan 2 buah data sebagai parameter, data pertama adalah string `"halo"` yang ditampung parameter `message`, dan parameter ke 2 adalah slice string `names` yang nilainya ditampung oleh parameter `arr`.

Di dalam `printMessage`, nilai `arr` yang merupakan slice string digabungkan menjadi sebuah string dengan pembatas adalah karakter **spasi**. Penggabungan slice dapat dilakukan dengan memanfaatkan fungsi `strings.Join()` (berada di dalam package `strings`).

```
[novalagung:belajar-golang $ go run bab17.go
halo John Wick
novalagung:belajar-golang $ ]
```

### A.17.2. Fungsi Dengan Return Value / Nilai Balik

Sebuah fungsi bisa didesain tidak mengembalikan nilai balik (*void*), atau bisa mengembalikan suatu nilai. Fungsi yang memiliki nilai kembalian, harus ditentukan tipe data nilai baliknya pada saat deklarasi.

Program berikut merupakan contoh penerapan fungsi yang memiliki return value.

```
package main

import (
    "fmt"
    "math/rand"
    "time"
)

func main() {
    rand.Seed(time.Now().Unix())
    var randomValue int

    randomValue = randomInRange(2, 10)
    fmt.Println("random number:", randomValue)
    randomValue = randomInRange(2, 10)
    fmt.Println("random number:", randomValue)
    randomValue = randomInRange(2, 10)
    fmt.Println("random number:", randomValue)
}

func randomInRange(min, max int) int {
    var value = rand.Int() % (max - min + 1) + min
    return value
}
```

Fungsi `randomInRange` bertugas untuk *generate* angka acak sesuai dengan range yang ditentukan, yang kemudian angka tersebut dijadikan nilai kembalian fungsi.

```
[novalagung:belajar-golang $ go run bab17.go
random number: 9
random number: 6
random number: 2
novalagung:belajar-golang $ ]
```

Cara menentukan tipe data nilai balik fungsi adalah dengan menuliskan tipe data yang diinginkan setelah kurung parameter. Bisa dilihat pada kode di atas, bahwa `int` merupakan tipe data nilai balik fungsi `randomInRange`.

```
func randomInRange(min, max int) int
```

Sedangkan cara untuk mengembalikan nilai itu sendiri adalah dengan menggunakan keyword `return` diikuti data yang ingin dikembalikan. Pada contoh di atas, `return value` artinya nilai variabel `value` dijadikan nilai kembalian fungsi.

Eksekusi keyword `return` akan menjadikan proses dalam blok fungsi berhenti pada saat itu juga. Semua statement setelah keyword tersebut tidak akan dieksekusi.

---

Dari kode di atas mungkin ada beberapa hal yang belum pernah kita lakukan pada bab-bab sebelumnya, kita akan bahas satu-persatu.

### A.17.3. Penggunaan Fungsi `rand.Seed()`

Fungsi ini diperlukan untuk memastikan bahwa angka random yang akan di-generate benar-benar acak. Kita bisa gunakan angka apa saja sebagai nilai parameter fungsi ini (umumnya diisi `time.Now().Unix()`).

```
rand.Seed(time.Now().Unix())
```

Fungsi `rand.Seed()` berada dalam package `math/rand`, yang harus di-import terlebih dahulu sebelum bisa dimanfaatkan.

Package `time` juga perlu di-import karena kita menggunakan fungsi `(time.Now().Unix())` disitu.

## A.17.4. Import Banyak Package

Penulisan keyword `import` untuk banyak package bisa dilakukan dengan dua cara, dengan menuliskannya di tiap package, atau cukup sekali saja, bebas.

```
import "fmt"
import "math/rand"
import "time"

// atau

import (
    "fmt"
    "math/rand"
    "time"
)
```

## A.17.5. Deklarasi Parameter Bertipe Data Sama

Khusus untuk fungsi yang tipe data parameternya sama, bisa ditulis dengan gaya yang unik. Tipe datanya dituliskan cukup sekali saja di akhir. Contohnya bisa dilihat pada kode berikut.

```
func nameOfFunc(paramA type, paramB type, paramC type) returnType
func nameOfFunc(paramA, paramB, paramC type) returnType

func randomInRange(min int, max int) int
func randomInRange(min, max int) int
```

## A.17.6. Penggunaan Keyword `return` Untuk Menghentikan Proses Dalam Fungsi

Selain sebagai penanda nilai balik, keyword `return` juga bisa dimanfaatkan untuk menghentikan proses dalam blok fungsi dimana ia dipakai. Contohnya bisa dilihat pada kode berikut.

```
package main

import "fmt"

func main() {
    divideNumber(10, 2)
    divideNumber(4, 0)
    divideNumber(8, -4)
}

func divideNumber(m, n int) {
    if n == 0 {
        fmt.Printf("invalid divider. %d cannot divided by %d\n", m, n)
        return
    }

    var res = m / n
```

```
    fmt.Printf("%d / %d = %d\n", m, n, res)
}
```

Fungsi `divideNumber` didesain tidak memiliki nilai balik. Fungsi ini dibuat untuk membungkus proses pembagian 2 bilangan, lalu menampilkan hasilnya.

Dalamnya terdapat proses validasi nilai variabel pembagi, jika nilainya adalah 0, maka akan ditampilkan pesan bahwa pembagian tidak bisa dilakukan, lalu proses dihentikan pada saat itu juga (dengan memanfaatkan keyword `return`). Jika nilai pembagi valid, maka proses pembagian diteruskan.

```
[novalagung:belajar-golang $ go run bab17.go
10 / 2 = 5
invalid divider. 4 cannot divided by 0
8 / -4 = -2
novalagung:belajar-golang $ ]
```

## A.18. Fungsi Multiple Return

Umumnya fungsi hanya memiliki satu buah nilai balik saja. Jika ada kebutuhan dimana data yang dikembalikan harus banyak, biasanya digunakanlah tipe seperti `map`, `slice`, atau `struct` sebagai nilai balik.

Golang menyediakan kapabilitas bagi programmer untuk membuat fungsi memiliki banyak nilai balik. Di bab ini akan dibahas bagaimana penerapannya.

### A.18.1 Penerapan Fungsi Multiple Return

Cara membuat fungsi yang memiliki banyak nilai balik tidaklah sulit. Tinggal tulis saja pada saat deklarasi fungsi semua tipe data nilai yang dikembalikan, dan pada keyword `return` tulis semua data yang ingin dikembalikan. Contoh bisa dilihat pada berikut.

```
package main

import "fmt"
import "math"

func calculate(d float64) (float64, float64) {
    // hitung luas
    var area = math.Pi * math.Pow(d / 2, 2)
    // hitung keliling
    var circumference = math.Pi * d

    // kembalikan 2 nilai
    return area, circumference
}
```

Fungsi `calculate()` di atas menerima satu buah parameter (`diameter`) yang digunakan dalam proses perhitungan. Di dalam fungsi tersebut ada 2 hal yang dihitung, yaitu nilai **keliling** dan **lingkaran**. Kedua nilai tersebut kemudian dijadikan sebagai return value fungsi.

Cara pendefinisian banyak nilai balik bisa dilihat pada kode di atas, langsung tulis tipe data semua nilai balik dipisah tanda koma, lalu ditambahkan kurung diantaranya.

```
func calculate(d float64) (float64, float64)
```

Tak lupa di bagian penulisan keyword `return` harus dituliskan juga semua data yang dijadikan nilai balik (dengan pemisah tanda koma).

```
return area, circumference
```

Implementasi dari fungsi `calculate()` di atas, bisa dilihat pada kode berikut.

```
func main() {
    var diameter float64 = 15
    var area, circumference = calculate(diameter)

    fmt.Printf("luas lingkaran\t: %.2f \n", area)
    fmt.Printf("keliling lingkaran\t: %.2f \n", circumference)
}
```

Output program:

```
[novalagung:belajar-golang $ go run bab18.go
luas lingkaran      : 176.71
keliling lingkaran  : 47.12
novalagung:belajar-golang $ ]
```

Karena fungsi tersebut memiliki banyak nilai balik, maka pada pemanggilannya harus disiapkan juga banyak variabel untuk menampung nilai kembalian yang ada (sesuai jumlah nilai balik fungsi).

```
var area, circumference = calculate(diameter)
```

## A.18.2 Fungsi Dengan Predefined Return Value

Keunikan lainnya yang jarang ditemui di bahasa lain adalah, di Golang variabel yang digunakan sebagai nilai balik bisa didefinisikan di-awal.

```
func calculate(d float64) (area float64, circumference float64) {
    area = math.Pi * math.Pow(d / 2, 2)
    circumference = math.Pi * d

    return
}
```

Fungsi `calculate` kita modif menjadi lebih sederhana. Bisa dilihat di kode di atas, ada cukup banyak perbedaan dibanding fungsi `calculate` sebelumnya. Perhatikan kode berikut.

```
func calculate(d float64) (area float64, circumference float64) {
```

Fungsi dideklarasikan memiliki 2 buah tipe data, dan variabel yang nantinya dijadikan nilai balik juga dideklarasikan. Variabel `area` yang bertipe `float64`, dan `circumference` bertipe `float64`.

Karena variabel nilai balik sudah ditentukan di awal, untuk mengembalikan nilai cukup dengan memanggil `return` tanpa perlu diikuti variabel apapun. Nilai terakhir `area` dan `circumference` sebelum pemanggilan keyword `return` adalah hasil dari fungsi di atas.

---

Ada beberapa hal baru dari kode di atas yang perlu dibahas, seperti `math.Pow()` dan `math.Pi`. Berikut adalah penjelasannya.

## A.18.3. Penggunaan Fungsi `math.Pow()`

Fungsi `math.Pow()` digunakan untuk memangkat nilai. `math.Pow(2, 3)` berarti 2 pangkat 3, hasilnya 8. Fungsi ini berada dalam package `math`.

## A.18.4. Penggunaan Konstanta `math.Pi`

`math.Pi` adalah konstanta bawaan package `math` yang merepresentasikan **Pi** atau **22/7**.

## A.19. Fungsi Variadic

Golang mengadopsi konsep **variadic function** atau pembuatan fungsi dengan parameter sejenis yang tak terbatas. Maksud **tak terbatas** disini adalah jumlah parameter yang disisipkan ketika pemanggilan fungsi bisa berapa saja.

Parameter variadic memiliki sifat yang mirip dengan slice. Nilai dari parameter-parameter yang disisipkan bertipe data sama, dan ditampung oleh sebuah variabel saja. Cara pengaksesan tiap datanya juga sama, dengan menggunakan index.

Di bab ini kita akan belajar mengenai cara penerapan fungsi variadic.

### A.19.1. Penerapan Fungsi Variadic

Deklarasi parameter variadic sama dengan cara deklarasi variabel biasa, pembedanya adalah pada parameter jenis ini ditambahkan tanda titik tiga kali ( ... ) tepat setelah penulisan variabel (sebelum tipe data). Nantinya semua nilai yang disisipkan sebagai parameter akan ditampung oleh variabel tersebut.

Berikut merupakan contoh peniterepannya.

```
package main

import "fmt"

func main() {
    var avg = calculate(2, 4, 3, 5, 4, 3, 3, 5, 5, 3)
    var msg = fmt.Sprintf("Rata-rata : %.2f", avg)
    fmt.Println(msg)
}

func calculate(numbers ...int) float64 {
    var total int = 0
    for _, number := range numbers {
        total += number
    }

    var avg = float64(total) / float64(len(numbers))
    return avg
}
```

Output program:

```
[novalagung:belajar-golang $ go run bab19.go
Rata-rata : 3.70
novalagung:belajar-golang $ ]
```

Bisa dilihat pada fungsi `calculate()`, parameter `numbers` dideklarasikan dengan disisipkan tanda 3 titik ( ... ), menandakan bahwa `numbers` adalah sebuah parameter variadic dengan tipe data `int`.

```
func calculate(numbers ...int) float64 {
```

Pemanggilan fungsi dilakukan seperti biasa, hanya saja jumlah parameter yang disisipkan bisa banyak.

```
var avg = calculate(2, 4, 3, 5, 4, 3, 3, 5, 5, 3)
```

Nilai tiap parameter bisa diakses seperti cara pengaksesan tiap elemen slice. Pada contoh di atas metode yang dipilih adalah `for - range`.

```
for _, number := range numbers {
```

Berikut merupakan penjelasan tambahan dari kode yang telah kita tulis.

## A.19.2. Penggunaan Fungsi `fmt.Sprintf()`

Fungsi `fmt.Sprintf()` pada dasarnya sama dengan `fmt.Printf()`, hanya saja fungsi ini tidak menampilkan nilai, melainkan mengembalikan nilainya dalam bentuk string. Pada kasus di atas, nilai kembalian `fmt.Sprintf()` ditampung oleh variabel `msg`.

Selain `fmt.Sprintf()`, ada juga `fmt.Sprint()` dan `fmt.Sprintln()`.

## A.19.3. Penggunaan Fungsi `float64()`

Sebelumnya sudah dibahas bahwa `float64` merupakan tipe data. Tipe data jika ditulis sebagai fungsi (penandanya ada tanda kurungnya) berguna untuk **casting**. Casting sendiri adalah teknik untuk konversi tipe sebuah data ke tipe lain. Hampir semua jenis tipe data dasar yang telah dipelajari di bab 9 bisa digunakan untuk casting. Dan cara penerepannya juga sama, cukup panggil sebagai fungsi, lalu masukan data yang ingin dikonversi sebagai parameter.

Pada contoh di atas, variabel `total` yang tipenya adalah `int`, dikonversi menjadi `float64`, begitu juga `len(numbers)` yang menghasilkan `int` dikonversi ke `float64`.

Variabel `avg` perlu dijadikan `float64` karena penghitungan rata-rata lebih sering menghasilkan nilai desimal.

Operasi bilangan (perkalian, pembagian, dan lainnya) di Golang hanya bisa dilakukan jika tipe datanya sejenis. Maka dari itulah perlu adanya casting ke tipe `float64` pada tiap operand.

## A.19.4. Pengisian Parameter Fungsi Variadic Menggunakan Data Slice

Slice bisa digunakan sebagai parameter variadic. Caranya dengan menambahkan tanda titik tiga kali, tepat setelah nama variabel yang dijadikan parameter. Contohnya bisa dilihat pada kode berikut.

```
var numbers = []int{2, 4, 3, 5, 4, 3, 3, 5, 5, 3}
var avg = calculate(numbers...)
var msg = fmt.Sprintf("Rata-rata : %.2f", avg)

fmt.Println(msg)
```

Pada kode di atas, variabel `numbers` yang merupakan slice int, disisipkan ke fungsi `calculate()` sebagai parameter variadic (bisa dilihat tanda 3 titik setelah penulisan variabel). Teknik ini sangat berguna ketika sebuah data slice ingin difungsikan sebagai parameter variadic.

Perhatikan juga kode berikut ini. Intinya adalah sama, hanya caranya yang berbeda.

```
var numbers = []int{2, 4, 3, 5, 4, 3, 3, 5, 5, 3}
var avg = calculate(numbers...)

// atau
```

```
var avg = calculate(2, 4, 3, 5, 4, 3, 3, 5, 5, 3)
```

Pada deklarasi parameter fungsi variadic, tanda 3 titik ( ...) dituliskan sebelum tipe data parameter. Sedangkan pada pemanggilan fungsi dengan menyisipkan parameter array, tanda tersebut dituliskan dibelakang variabelnya.

## A.19.5. Fungsi Dengan Parameter Biasa & Variadic

Parameter variadic bisa dikombinasikan dengan parameter biasa, dengan syarat parameter variadic-nya harus diposisikan di akhir. Contohnya bisa dilihat pada kode berikut.

```
import "fmt"
import "strings"

func yourHobbies(name string, hobbies ...string) {
    var hobbiesAsString = strings.Join(hobbies, ", ")

    fmt.Printf("Hello, my name is: %s\n", name)
    fmt.Printf("My hobbies are: %s\n", hobbiesAsString)
}
```

Nilai parameter pertama fungsi `yourHobbies()` akan ditampung oleh `name`, sedangkan nilai parameter kedua dan seterusnya akan ditampung oleh `hobbies` sebagai slice.

Cara pemanggilannya masih sama seperti pada fungsi biasa.

```
func main() {
    yourHobbies("wick", "sleeping", "eating")
}
```

Jika parameter kedua dan seterusnya ingin diisi dengan data dari slice, maka gunakan tanda titik tiga kali.

```
func main() {
    var hobbies = []string{"sleeping", "eating"}
    yourHobbies("wick", hobbies...)
}
```

Output program:

```
[novalagung:belajar-golang $ go run bab19.go
Hello, my name is: wick
My hobbies are: sleeping, eating
novalagung:belajar-golang $ ]
```

## A.20. Fungsi Closure

Definisi **Closure** adalah sebuah fungsi yang bisa disimpan dalam variabel. Dengan menerapkan konsep tersebut, kita bisa membuat fungsi didalam fungsi, atau bahkan membuat fungsi yang mengembalikan fungsi.

Closure merupakan *anonymous function* atau fungsi tanpa nama. Biasa dimanfaatkan untuk membungkus suatu proses yang hanya dipakai sekali atau dipakai pada blok tertentu saja.

### A.20.1. Closure Disimpan Sebagai Variabel

Sebuah fungsi tanpa nama bisa disimpan dalam variabel. Variabel yang menyimpan closure memiliki sifat seperti fungsi yang disimpannya. Di bawah ini adalah contoh program sederhana untuk mencari nilai terendah dan tertinggi dari suatu array. Logika pencarian dibungkus dalam closure yang ditampung oleh variabel `getMinMax`.

```
package main

import "fmt"

func main() {
    var getMinMax = func(n []int) (int, int) {
        var min, max int
        for i, e := range n {
            switch {
            case i == 0:
                max, min = e, e
            case e > max:
                max = e
            case e < min:
                min = e
            }
        }
        return min, max
    }

    var numbers = []int{2, 3, 4, 3, 4, 2, 3}
    var min, max = getMinMax(numbers)
    fmt.Printf("data : %v\nmin : %v\nmax : %v\n", numbers, min, max)
}
```

Bisa dilihat pada kode di atas bagaimana sebuah closure dibuat dan dipanggil. Sedikit berbeda memang dibanding pembuatan fungsi biasa. Fungsi ditulis tanpa nama, lalu ditampung dalam variabel.

```
var getMinMax = func(n []int) (int, int) {
    // ...
}
```

Cara pemanggilannya, dengan menuliskan nama variabel tersebut sebagai fungsi, seperti pemanggilan fungsi biasa.

```
var min, max = getMinMax(numbers)
```

Output program:

```
[novalagung:belajar-golang $ go run bab20.go
data : [2 3 4 3 4 2 3]
min : 2
max : 4
novalagung:belajar-golang $ ]
```

Berikut adalah penjelasan tambahan mengenai kode di atas

## A.20.2. Penggunaan Template String %v

Template `%v` digunakan untuk menampilkan segala jenis data. Bisa array, int, float, bool, dan lainnya.

```
fmt.Printf("data : %v\nmin : %v\nmax : %v\n", numbers, min, max)
```

Bisa dilihat pada statement di atas, data bertipe array dan numerik ditampilkan menggunakan `%v`. Template ini biasa dimanfaatkan untuk menampilkan sebuah data yang tipe nya bisa dinamis atau belum diketahui. Sangat tepat jika digunakan pada data bertipe `interface{}` yang nantinya akan di bahas pada bab 27.

## A.20.3. Immediately-Invoked Function Expression (IIFE)

Closure jenis ini dieksekusi langsung pada saat deklarasi nya. Biasa digunakan untuk membungkus proses yang hanya dilakukan sekali, bisa mengembalikan nilai, bisa juga tidak.

Di bawah ini merupakan contoh sederhana penerapan metode IIFE untuk filtering data array.

```
package main

import "fmt"

func main() {
    var numbers = []int{2, 3, 0, 4, 3, 2, 0, 4, 2, 0, 3}

    var newNumbers = func(min int) []int {
        var r []int
        for _, e := range numbers {
            if e < min {
                continue
            }
            r = append(r, e)
        }
        return r
    }(3)

    fmt.Println("original number :", numbers)
    fmt.Println("filtered number :", newNumbers)
}
```

Output program:

```
[novalagung:belajar-golang $ go run bab20.go
original number : [2 3 0 4 3 2 0 4 2 0 3]
filtered number : [3 4 3 4 3]
novalagung:belajar-golang $ ]
```

Ciri khas IIFE adalah adanya kurung parameter tepat setelah deklarasi closure berakhir. Jika ada parameter, bisa juga dituliskan dalam kurung parameternya.

```
var newNumbers = func(min int) []int {
    // ...
}(3)
```

Pada contoh di atas IIFE menghasilkan nilai balik yang kemudian ditampung `newNumber`. Perlu diperhatikan bahwa yang ditampung adalah **nilai kembalinya** bukan body fungsi atau **closure**.

Closure bisa juga dengan gaya manifest typing, caranya dengan menuliskan skema closure-nya sebagai tipe data. Contoh:

```
var closure (func (string, int, []string) int)
closure = func (a string, b int, c []string) int {
    // ...
}
```

## A.20.4. Closure Sebagai Nilai Kembalian

Salah satu keunikan closure lainnya adalah bisa dijadikan sebagai nilai balik fungsi, cukup aneh memang, tapi pada suatu kondisi teknik ini sangat membantu. Di bawah ini disediakan sebuah fungsi bernama `findMax()`, fungsi ini salah satu nilai kembalinya berupa closure.

```
package main

import "fmt"

func findMax(numbers []int, max int) (int, func() []int) {
    var res []int
    for _, e := range numbers {
        if e <= max {
            res = append(res, e)
        }
    }
    return len(res), func() []int {
        return res
    }
}
```

Nilai kembalian ke-2 pada fungsi di atas adalah closure dengan skema `func() []int`. Bisa dilihat di bagian akhir, ada fungsi tanpa nama yang dikembalikan.

```
return len(res), func() []int {
    return res
}
```

Fungsi tanpa nama yang akan dikembalikan boleh disimpan pada variabel terlebih dahulu. Contohnya:

```
var getNumbers = func() []int {
    return res
}
return len(res), getNumbers
```

Sedikit tentang fungsi `findMax()`, fungsi ini digunakan untuk mencari banyaknya angka-angka yang nilainya di bawah atau sama dengan angka tertentu. Nilai kembalian pertama adalah jumlah angkanya. Nilai kembalian kedua berupa closure yang mengembalikan angka-angka yang dicari. Berikut merupakan contoh implementasi fungsi tersebut.

```
func main() {
    var max = 3
    var numbers = []int{2, 3, 0, 4, 3, 2, 0, 4, 2, 0, 3}
    var howMany, getNumbers = findMax(numbers, max)
    var theNumbers = getNumbers()

    fmt.Println("numbers\t:", numbers)
    fmt.Printf("find \t: %d\n\n", max)

    fmt.Println("found \t:", howMany) // 9
```

```
    fmt.Println("value \t:", theNumbers) // [2 3 0 3 2 0 2 0 3]
}
```

Output program:

```
[novalagung:belajar-golang $ go run bab20.go
numbers : [2 3 0 4 3 2 0 4 2 0 3]
find     : 3
found    : 9
value    : [2 3 0 3 2 0 2 0 3]
novalagung:belajar-golang $ ]
```

## A.21. Fungsi Sebagai parameter

Setelah di bab sebelumnya kita belajar mengenai fungsi yang mengembalikan nilai balik berupa fungsi, kali ini topiknya tidak kalah unik, yaitu fungsi yang digunakan sebagai parameter.

Di Golang, fungsi bisa dijadikan sebagai tipe data variabel. Dari situ sangat memungkinkan untuk menjadikannya sebagai parameter juga.

### A.21.1. Penerapan Fungsi Sebagai Parameter

Cara membuat parameter fungsi adalah dengan langsung menuliskan skema fungsi nya sebagai tipe data. Contohnya bisa dilihat pada kode berikut.

```
package main

import "fmt"
import "strings"

func filter(data []string, callback func(string) bool) []string {
    var result []string
    for _, each := range data {
        if filtered := callback(each); filtered {
            result = append(result, each)
        }
    }
    return result
}
```

Parameter `callback` merupakan sebuah closure yang dideklarasikan bertipe `func(string) bool`. Closure tersebut dipanggil di tiap perulangan dalam fungsi `filter()`.

Fungsi `filter()` sendiri kita buat untuk filtering data array (yang datanya didapat dari parameter pertama), dengan kondisi filter bisa ditentukan sendiri. Di bawah ini adalah contoh pemanfaatan fungsi tersebut.

```
func main() {
    var data = []string{"wick", "jason", "ethan"}
    var dataContains0 = filter(data, func(each string) bool {
        return strings.Contains(each, "o")
    })
    var dataLength5 = filter(data, func(each string) bool {
        return len(each) == 5
    })

    fmt.Println("data asli \t\t:", data)
    // data asli : [wick jason ethan]

    fmt.Println("filter ada huruf \"o\"\t\t:", dataContains0)
    // filter ada huruf "o" : [jason]

    fmt.Println("filter jumlah huruf \"5\"\t\t:", dataLength5)
    // filter jumlah huruf "5" : [jason ethan]
}
```

Ada cukup banyak hal yang terjadi didalam tiap pemanggilan fungsi `filter()` di atas. Berikut merupakan penjelasannya.

1. Data array (yang didapat dari parameter pertama) akan di-looping.

2. Di tiap perulangannya, closure `callback` dipanggil, dengan disisipkan data tiap elemen perulangan sebagai parameter.
3. Closure `callback` berisikan kondisi filtering, dengan hasil bertipe `bool` yang kemudian dijadikan nilai balik dikembalikan.
4. Di dalam fungsi `filter()` sendiri, ada proses seleksi kondisi (yang nilainya didapat dari hasil eksekusi closure `callback`). Ketika kondisinya bernilai `true`, maka data elemen yang sedang diulang dinyatakan lolos proses filtering.
5. Data yang lolos ditampung variabel `result`. Variabel tersebut dijadikan sebagai nilai balik fungsi `filter()`.

```
[novalagung:belajar-golang $ go run bab21.go
data asli           : [wick jason ethan]
filter ada huruf "o" : [jason]
filter jumlah huruf "5" : [jason ethan]
novalagung:belajar-golang $ ]
```

Pada `dataContains0`, parameter kedua fungsi `filter()` berisikan statement untuk deteksi apakah terdapat substring "`o`" di dalam nilai variabel `each` (yang merupakan data tiap elemen), jika iya, maka kondisi filter bernilai `true`, dan sebaliknya.

pada contoh ke-2 (`dataLength5`), closure `callback` berisikan statement untuk deteksi jumlah karakter tiap elemen. Jika ada elemen yang jumlah karakternya adalah 5, berarti elemen tersebut lolos filter.

Memang butuh usaha ekstra untuk memahami pemanfaatan closure sebagai parameter fungsi. Tapi setelah paham, penerapan teknik ini pada kondisi yang tepat akan sangat membantu proses pembuatan aplikasi.

## A.21.2. Alias Skema Closure

Kita sudah mempelajari bahwa closure bisa dimanfaatkan sebagai tipe parameter, contohnya seperti pada fungsi `filter()`. Pada fungsi tersebut kebetulan skema tipe parameter closure-nya tidak terlalu panjang, hanya ada satu buah parameter dan satu buah nilai balik.

Pada fungsi yang skema-nya cukup panjang, akan lebih baik jika menggunakan alias, apalagi ketika ada parameter fungsi lain yang juga menggunakan skema yang sama. Membuat alias fungsi berarti menjadikan skema fungsi tersebut menjadi tipe data baru. Caranya dengan menggunakan keyword `type`. Contoh:

```
type FilterCallback func(string) bool

func filter(data []string, callback FilterCallback) []string {
    // ...
}
```

Skema `func(string) bool` diubah menjadi tipe dengan nama `FilterCallback`. Tipe tersebut kemudian digunakan sebagai tipe data parameter `callback`.

---

Di bawah ini merupakan penjelasan tambahan mengenai fungsi `strings.Contains()`.

## A.21.3. Penggunaan Fungsi `string.Contains()`

Inti dari fungsi ini adalah untuk deteksi apakah sebuah substring adalah bagian dari string, jika iya maka akan bernilai `true`, dan sebaliknya. Contoh penggunaannya:

```
var result = strings.Contains("Golang", "ang")
// true
```

Variabel `result` bernilai `true` karena string `"ang"` merupakan bagian dari string `"Golang"`.

## A.22. Pointer

Pointer adalah *reference* atau alamat memory. Variabel pointer berarti variabel yang berisi alamat memori suatu nilai. Sebagai contoh sebuah variabel bertipe integer memiliki nilai **4**, maka yang dimaksud pointer adalah **alamat memori dimana nilai 4 disimpan**, bukan nilai 4 nya sendiri.

Variabel-variabel yang memiliki *reference* atau alamat pointer yang sama, saling berhubungan satu sama lain dan nilainya pasti sama. Ketika ada perubahan nilai, maka akan memberikan efek kepada variabel lain (yang referensinya sama) yaitu nilainya ikut berubah.

### A.22.1. Penerapan Pointer

Variabel bertipe pointer ditandai dengan adanya tanda **asterisk** ( `*` ) tepat sebelum penulisan tipe data ketika deklarasi.

```
var number *int
var name *string
```

Nilai default variabel pointer adalah `nil` (kosong). Variabel pointer tidak bisa menampung nilai yang bukan pointer, dan sebaliknya variabel biasa tidak bisa menampung nilai pointer.

Variabel biasa sebenarnya juga bisa diambil nilai pointernya, caranya dengan menambahkan tanda **ampersand** ( `&` ) tepat sebelum nama variabel. Metode ini disebut dengan **referencing**.

Dan sebaliknya, nilai asli variabel pointer juga bisa diambil, dengan cara menambahkan tanda **asterisk** ( `*` ) tepat sebelum nama variabel. Metode ini disebut dengan **dereferencing**.

OK, langsung saja kita praktikan. Berikut adalah contoh penerapan pointer.

```
var numberA int = 4
var numberB *int = &numberA

fmt.Println("numberA (value)    :", numberA) // 4
fmt.Println("numberA (address) : ", &numberA) // 0xc20800a220

fmt.Println("numberB (value)    :", *numberB) // 4
fmt.Println("numberB (address) : ", numberB) // 0xc20800a220
```

Variabel `numberB` dideklarasikan bertipe pointer `int` dengan nilai awal adalah referensi variabel `numberA` (bisa dilihat pada kode `&numberA`). Dengan ini, variabel `numberA` dan `numberB` menampung data dengan referensi alamat memori yang sama.

```
[novalagung:belajar-golang $ go run bab22.go
numberA (value)    : 4
numberA (address) : 0xc20800a220
numberB (value)    : 4
numberB (address) : 0xc20800a220
novalagung:belajar-golang $ ]
```

Variabel pointer jika di-print akan menghasilkan string alamat memori (dalam notasi heksadesimal), contohnya seperti `numberB` yang diprint menghasilkan `0xc20800a220`.

Nilai asli sebuah variabel pointer bisa didapatkan dengan cara di-dereference terlebih dahulu (bisa dilihat pada kode `*numberB`).

## A.22.2. Efek Perubahan Nilai Pointer

Ketika salah satu variabel pointer di ubah nilainya, sedang ada variabel lain yang memiliki referensi memori yang sama, maka nilai variabel lain tersebut juga akan berubah.

```
var numberA int = 4
var numberB *int = &numberA

fmt.Println("numberA (value)    :", numberA)
fmt.Println("numberA (address) :", &numberA)
fmt.Println("numberB (value)    :", *numberB)
fmt.Println("numberB (address) :", numberB)

fmt.Println("")

numberA = 5

fmt.Println("numberA (value)    :", numberA)
fmt.Println("numberA (address) :", &numberA)
fmt.Println("numberB (value)    :", *numberB)
fmt.Println("numberB (address) :", numberB)
```

Variabel `numberA` dan `numberB` memiliki referensi memori yang sama. Perubahan pada salah satu nilai variabel tersebut akan memberikan efek pada variabel lainnya. Pada contoh di atas, `numberA` nilainya di ubah menjadi `5`. membuat nilai asli variabel `numberB` ikut berubah menjadi `5`.

```
[novalagung:belajar-golang $ go run bab22.go
numberA (value)    : 4
numberA (address) : 0xc20800a220
numberB (value)    : 4
numberB (address) : 0xc20800a220

numberA (value)    : 5
numberA (address) : 0xc20800a220
numberB (value)    : 5
numberB (address) : 0xc20800a220
novalagung:belajar-golang $ ]
```

## A.22.3. Parameter Pointer

Parameter bisa juga didesain sebagai pointer. Cara penerapannya kurang lebih sama, dengan cara mendeklarasikan parameter sebagai pointer.

```
package main

import "fmt"

func main() {
    var number = 4
    fmt.Println("before :", number) // 4

    change(&number, 10)
    fmt.Println("after  :", number) // 10
}

func change(original *int, value int) {
    *original = value
}
```

Fungsi `change()` memiliki 2 parameter, yaitu `original` yang tipenya adalah pointer `int`, dan `value` yang bertipe `int`. Di dalam fungsi tersebut nilai asli parameter pointer `original` diubah.

Fungsi `change()` kemudian diimplementasikan di `main`. Variabel `number` yang nilai awalnya adalah `4` diambil referensi-nya lalu digunakan sebagai parameter pada pemanggilan fungsi `change()`.

Nilai variabel `number` berubah menjadi `10` karena perubahan yang terjadi di dalam fungsi `change` adalah pada variabel pointer.

```
[novalagung:belajar-golang $ go run bab22.go
before : 4
after  : 10
novalagung:belajar-golang $ ]
```

## A.23. Struct

Struct adalah kumpulan definisi variabel (atau property) dan atau fungsi (atau method), yang dibungkus dengan nama tertentu.

Property dalam struct, tipe datanya bisa bervariasi. Mirip seperti `map`, hanya saja key-nya sudah didefinisikan di awal, dan tipe data tiap itemnya bisa berbeda.

Dengan memanfaatkan struct, data akan terbungkus lebih rapi dan mudah di-maintain.

Struct merupakan cetakan, digunakan untuk mencetak variabel objek (istilah untuk variabel yang memiliki property). Variabel objek memiliki behaviour atau sifat yang sama sesuai struct pencetaknya. Konsep ini sama dengan konsep `class` pada pemrograman berbasis objek. Sebuah buah struct bisa dimanfaatkan untuk mencetak banyak objek.

Disini penulis menggunakan konsep OOP sebagai analogi, dengan tujuan untuk mempermudah dalam mencerna isi bab ini.

### A.23.1. Deklarasi Struct

Keyword `type` digunakan untuk deklarasi struct. Di bawah ini merupakan contoh cara penggunaannya.

```
type student struct {
    name string
    grade int
}
```

Struct `student` dideklarasikan memiliki 2 property, yaitu `name` dan `grade`. Objek yang dicetak dengan struct ini nantinya akan memiliki sifat yang sama.

### A.23.2. Penerapan Struct

Struct `student` yang sudah disiapkan di atas akan kita manfaatkan untuk mencetak variabel objek. Property variabel tersebut diisi kemudian ditampilkan.

```
func main() {
    var s1 student
    s1.name = "john wick"
    s1.grade = 2

    fmt.Println("name :", s1.name)
    fmt.Println("grade :", s1.grade)
}
```

Cara membuat variabel objek sama seperti pembuatan variabel biasa. Tinggal tulis saja nama variabel diikuti nama struct, contoh: `var s1 student`.

Semua property variabel objek pada awalnya memiliki nilai default sesuai tipe datanya.

Property variabel objek bisa diakses nilainya menggunakan notasi titik, contohnya `s1.name`. Nilai property-nya juga bisa diubah, contohnya `s1.grade = 2`.

```
[novalagung:belajar-golang $ go run bab23.go
name : john wick
grade : 2
novalagung:belajar-golang $ ]
```

### A.23.3. Inisialisasi Object Struct

Cara inisialisasi variabel objek adalah dengan menambahkan kurung kurawal setelah nama struct. Nilai masing-masing property bisa diisi pada saat inisialisasi.

Pada contoh berikut, terdapat 3 buah variabel objek yang dideklarasikan dengan cara berbeda.

```
var s1 = student{}
s1.name = "wick"
s1.grade = 2

var s2 = student{"ethan", 2}

var s3 = student{name: "jason"}

fmt.Println("student 1 :", s1.name)
fmt.Println("student 2 :", s2.name)
fmt.Println("student 3 :", s3.name)
```

Pada kode di atas, variabel `s1` menampung objek cetakan `student`. Variabel tersebut kemudian di-set nilai property-nya.

Variabel objek `s2` dideklarasikan dengan metode yang sama dengan `s1`, pembedanya di `s2` nilai propertinya di isi langsung ketika deklarasi. Nilai pertama akan menjadi nilai property pertama (yaitu `name`), dan selanjutnya berurutan.

Pada deklarasi `s3`, dilakukan juga pengisian property ketika pencetakan objek. Hanya saja, yang diisi hanya `name` saja. Cara ini cukup efektif jika digunakan untuk membuat objek baru yang nilai property-nya tidak semua harus disiapkan di awal. Keistimewaan lain menggunakan cara ini adalah penentuan nilai property bisa dilakukan dengan tidak berurutan. Contohnya:

```
var s4 = student{name: "wayne", grade: 2}
var s5 = student{grade: 2, name: "bruce"}
```

### A.23.4. Variabel Objek Pointer

Objek hasil cetakan struct bisa diambil nilai pointer-nya, dan bisa disimpan pada variabel objek yang bertipe struct pointer. Contoh penerapannya:

```
var s1 = student{name: "wick", grade: 2}

var s2 *student = &s1
fmt.Println("student 1, name :", s1.name)
fmt.Println("student 4, name :", s2.name)

s2.name = "ethan"
fmt.Println("student 1, name :", s1.name)
fmt.Println("student 4, name :", s2.name)
```

`s2` adalah variabel pointer hasil cetakan struct `student`. `s2` menampung nilai referensi `s1`, menjadikan setiap perubahan pada property variabel tersebut, akan juga berpengaruh pada variabel objek `s1`.

Meskipun `s2` bukan variabel asli, property nya tetap bisa diakses seperti biasa. Inilah keistimewaan property dalam objek pointer, tanpa perlu di-dereferensi nilai asli property tetap bisa diakses. Pengisian nilai pada property tersebut juga bisa langsung menggunakan nilai asli, contohnya seperti `s2.name = "ethan"`.

```
[novalagung:belajar-golang $ go run bab23.go
student 1, name : wick
student 4, name : wick
student 1, name : ethan
student 4, name : ethan
novalagung:belajar-golang $ ]
```

## A.23.5. Embedded Struct

**Embedded** struct adalah mekanisme untuk menyimpan objek cetakan struct kedalam properti sebuah struct lain. Agar lebih mudah dipahami, mari kita bahas kode berikut.

```
package main

import "fmt"

type person struct {
    name string
    age int
}

type student struct {
    grade int
    person
}

func main() {
    var s1 = student{}
    s1.name = "wick"
    s1.age = 21
    s1.grade = 2

    fmt.Println("name :", s1.name)
    fmt.Println("age :", s1.age)
    fmt.Println("age :", s1.person.age)
    fmt.Println("grade :", s1.grade)
}
```

Pada kode di atas, disiapkan struct `person` dengan properti yang tersedia adalah `name` dan `age`. Disiapkan juga struct `student` dengan property `grade`. Struct `person` di-embed kedalam struct `student`. Caranya cukup mudah, yaitu dengan menuliskan nama struct yang ingin di-embed ke dalam body `struct` target.

Embedded struct adalah **mutable**, nilai property-nya bisa diubah.

Khusus untuk properti yang bukan properti asli (properti turunan dari struct lain), bisa diakses dengan cara mengakses struct *parent*-nya terlebih dahulu, contohnya `s1.person.age`. Nilai yang dikembalikan memiliki referensi yang sama dengan `s1.age`.

## A.23.6. Embedded Struct Dengan Nama Property Yang Sama

Jika salah satu nama properti sebuah struct memiliki kesamaan dengan properti milik struct lain yang di-embed, maka pengaksesan property-nya harus dilakukan secara explisit atau jelas. Contoh bisa dilihat di kode berikut.

```
package main
```

```

import "fmt"

type person struct {
    name string
    age  int
}

type student struct {
    person
    age   int
    grade int
}

func main() {
    var s1 = student{}
    s1.name = "wick"
    s1.age = 21          // age of student
    s1.person.age = 22 // age of person

    fmt.Println(s1.name)
    fmt.Println(s1.age)
    fmt.Println(s1.person.age)
}

```

Struct `person` di-embed ke dalam struct `student`, dan kedua struct tersebut kebetulan salah satu nama property-nya ada yg sama, yaitu `age`. Cara mengakses property `age` milik struct `person` lewat objek struct `student`, adalah dengan menuliskan nama struct yg di-embed kemudian nama property-nya, contohnya: `s1.person.age = 22`.

## A.23.7. Pengisian Nilai Sub-Struct

Pengisian nilai property sub-struct bisa dilakukan dengan langsung memasukkan variabel objek yang tercetak dari struct yang sama.

```

var p1 = person{name: "wick", age: 21}
var s1 = student{person: p1, grade: 2}

fmt.Println("name : ", s1.name)
fmt.Println("age  : ", s1.age)
fmt.Println("grade : ", s1.grade)

```

Pada deklarasi `s1`, property `person` diisi variabel objek `p1`.

## A.23.8. Anonymous Struct

Anonymous struct adalah struct yang tidak dideklarasikan di awal, melainkan ketika dibutuhkan saja, langsung pada saat penciptaan objek. Teknik ini cukup efisien untuk pembuatan variabel objek yang struct nya hanya dipakai sekali.

```

package main

import "fmt"

type person struct {
    name string
    age  int
}

func main() {
    var s1 = struct {
        person
        ...
    }
}
```

```

        grade int
    }{}
s1.person = person{"wick", 21}
s1.grade = 2

fmt.Println("name :", s1.person.name)
fmt.Println("age  :", s1.person.age)
fmt.Println("grade:", s1.grade)
}

```

Pada kode di atas, variabel `s1` langsung diisi objek anonymous struct yang memiliki property `grade`, dan property `person` yang merupakan embedded struct.

Salah satu aturan yang perlu diingat dalam pembuatan anonymous struct adalah, deklarasi harus diikuti dengan inisialisasi. Bisa dilihat pada `s1` setelah deklarasi struktur struct, terdapat kurung kurawal untuk inisialisasi objek. Meskipun nilai tidak diisiakan di awal, kurung kurawal tetap harus ditulis.

```

// anonymous struct tanpa pengisian property
var s1 = struct {
    person
    grade int
} {}

// anonymous struct dengan pengisian property
var s2 = struct {
    person
    grade int
} {
    person: person{"wick", 21},
    grade: 2,
}

```

## A.23.9. Kombinasi Slice & Struct

Slice dan `struct` bisa dikombinasikan seperti pada slice dan `map`, caranya pun mirip, cukup tambahkan tanda `[]` sebelum tipe data pada saat deklarasi.

```

type person struct {
    name string
    age  int
}

var allStudents = []person{
    {name: "Wick", age: 23},
    {name: "Ethan", age: 23},
    {name: "Bourne", age: 22},
}

for _, student := range allStudents {
    fmt.Println(student.name, "age is", student.age)
}

```

## A.23.10. Inisialisasi Slice Anonymous Struct

Anonymous struct bisa dijadikan sebagai tipe sebuah slice. Dan nilai awalnya juga bisa diinisialisasi langsung pada saat deklarasi. Berikut adalah contohnya:

```

var allStudents = []struct {
    person
}

```

```

grade int
} {
    {person: person{"wick", 21}, grade: 2},
    {person: person{"ethan", 22}, grade: 3},
    {person: person{"bond", 21}, grade: 3},
}

for _, student := range allStudents {
    fmt.Println(student)
}

```

## A.23.11. Deklarasi Anonymous Struct Menggunakan Keyword var

Cara lain untuk deklarasi anonymous struct adalah dengan menggunakan keyword `var`.

```

var student struct {
    person
    grade int
}

student.person = person{"wick", 21}
student.grade = 2

```

Statement `type student struct` adalah contoh cara deklarasi struct. Maknanya akan berbeda ketika keyword `type` diganti `var`, seperti pada contoh di atas `var student struct`, yang artinya dicetak sebuah objek dari anonymous struct kemudian disimpan pada variabel bernama `student`.

Deklarasi anonymous struct menggunakan metode ini juga bisa dilakukan beserta inisialisasi-nya.

```

// hanya deklarasi
var student struct {
    grade int
}

// deklarasi sekaligus inisialisasi
var student = struct {
    grade int
} {
    12,
}

```

## A.23.12. Nested struct

Nested struct adalah anonymous struct yang di-embed ke sebuah struct. Deklarasinya langsung didalam struct peng-embed. Contoh:

```

type student struct {
    person struct {
        name string
        age int
    }
    grade int
    hobbies []string
}

```

Teknik ini biasa digunakan ketika decoding data **json** yang strukturnya cukup kompleks dengan proses decode hanya sekali.

## A.23.13. Deklarasi Dan Inisialisasi Struct Secara Horizontal

Deklarasi struct bisa dituliskan secara horizontal, caranya bisa dilihat pada kode berikut:

```
type person struct { name string; age int; hobbies []string }
```

Tanda semi-colon ( ; ) digunakan sebagai pembatas deklarasi property yang dituliskan secara horizontal. Inisialisasi nilai juga bisa dituliskan dengan metode ini. Contohnya:

```
var p1 = struct { name string; age int } { age: 22, name: "wick" }
var p2 = struct { name string; age int } { "ethan", 23 }
```

Bagi pengguna editor Sublime yang terinstal plugin GoSublime didalamnya, cara ini tidak akan bisa dilakukan, karena setiap kali file di-save, kode program dirapikan. Jadi untuk mengetesnya bisa dengan menggunakan editor lain.

## A.23.14. Tag property dalam struct

Tag merupakan informasi opsional yang bisa ditambahkan pada masing-masing property struct.

```
type person struct {
    name string `tag1`
    age   int    `tag2`
}
```

Tag bisa dimanfaatkan untuk keperluan encode/decode data json. Informasi tag juga bisa diakses lewat reflect. Nantinya akan ada pembahasan yang lebih detail mengenai pemanfaatan tag dalam struct, terutama ketika sudah masuk bab JSON.

## A.23.15. Type Alias

Sebuah tipe data, seperti struct, bisa dibuatkan alias baru, caranya dengan `type NamaAlias = TargetStruct`. Contoh:

```
type Person struct {
    name string
    age  int
}
type People = Person

var p1 = Person{"wick", 21}
fmt.Println(p1)

var p2 = People{"wick", 21}
fmt.Println(p2)
```

Pada kode di atas, sebuah alias bernama `People` dibuat untuk struct `Person`.

Casting dari objek (yang dicetak lewat struct tertentu) ke tipe yang merupakan alias dari struct pencetak, hasilnya selalu valid. Berlaku juga sebaliknya.

```
people := People{"wick", 21}
```

```
fmt.Println(Person(people))

person := Person{"wick", 21}
fmt.Println(People(person))
```

Pembuatan struct baru juga bisa dilakukan lewat teknik type alias. Silakan perhatikan kode berikut.

```
type People1 struct {
    name string
    age  int
}

type People2 = struct {
    name string
    age  int
}
```

Struct `People1` dideklarasikan. Struct alias `People2` juga dideklarasikan, struct ini merupakan alias dari anonymous struct. Penggunaan teknik type alias untuk anonymous struct menghasilkan output yang ekuivalen dengan pendeklarasian struct.

Perlu diketahui juga, dan di atas sudah sempat disinggung, bahwa teknik type alias ini tidak didesain hanya untuk pembuatan alias pada tipe struct saja, semua jenis tipe data bisa dibuatkan alias. Di contoh berikut, dipersiapkan tipe `Number` yang merupakan alias dari tipe data `int`.

```
type Number = int
var num Number = 12
```

## A.24. Method

**Method** adalah fungsi yang menempel pada `struct`, sehingga hanya bisa diakses lewat variabel objek.

Keunggulan method dibanding fungsi biasa adalah memiliki akses ke property struct hingga level *private* (level akses nantinya akan dibahas lebih detail pada bab selanjutnya). Dan juga, dengan menggunakan method sebuah proses bisa di-enkapsulasi dengan baik.

### A.24.1. Penerapan Method

Cara menerapkan method sedikit berbeda dibanding penggunaan fungsi. Ketika deklarasi, ditentukan juga siapa pemilik method tersebut. Contohnya bisa dilihat pada kode berikut:

```
package main

import "fmt"
import "strings"

type student struct {
    name string
    grade int
}

func (s student) sayHello() {
    fmt.Println("halo", s.name)
}

func (s student) getNameAt(i int) string {
    return strings.Split(s.name, " ")[i-1]
}
```

Cara deklarasi method sama seperti fungsi, hanya saja perlu ditambahkan deklarasi variabel objek di sela-sela keyword `func` dan nama fungsi. Struct yang digunakan akan menjadi pemilik method.

`func (s student) sayHello()` maksudnya adalah fungsi `sayHello` dideklarasikan sebagai method milik struct `student`. Pada contoh di atas struct `student` memiliki dua buah method, yaitu `sayHello()` dan `getNameAt()`.

Contoh pemanfaatan method bisa dilihat pada kode berikut.

```
func main() {
    var s1 = student{"john wick", 21}
    s1.sayHello()

    var name = s1.getNameAt(2)
    fmt.Println("nama panggilan :", name)
}
```

Output:

```
[novalagung:belajar-golang $ go run bab24.go
halo john wick
nama panggilan : wick
novalagung:belajar-golang $ ]
```

Cara mengakses method sama seperti pengaksesan properti berupa variabel. Tinggal panggil saja methodnya.

```
s1.sayHello()
var name = s1.getNameAt(2)
```

Method memiliki sifat yang sama persis dengan fungsi biasa. Seperti bisa berparameter, memiliki nilai balik, dan lainnya. Dari segi sintaks, pembedanya hanya ketika pengaksesan dan deklarasi. Bisa dilihat di kode berikut, sekilas perbandingan penulisan fungsi dan method.

```
func sayHello() {
func (s student) sayHello() {

func getNameAt(i int) string {
func (s student) getNameAt(i int) string {
```

## A.24.2. Method Pointer

Method pointer adalah method yang variabel objek pemilik method tersebut berupa pointer.

Kelebihan method jenis ini adalah, ketika kita melakukan manipulasi nilai pada property lain yang masih satu struct, nilai pada property tersebut akan di rubah pada reference nya. Lebih jelasnya perhatikan kode berikut.

```
package main

import "fmt"

type student struct {
    name string
    grade int
}

func (s student) changeName1(name string) {
    fmt.Println("---> on changeName1, name changed to", name)
    s.name = name
}

func (*student) changeName2(name string) {
    fmt.Println("---> on changeName2, name changed to", name)
    s.name = name
}

func main() {
    var s1 = student{"john wick", 21}
    fmt.Println("s1 before", s1.name)
    // john wick

    s1.changeName1("jason bourne")
    fmt.Println("s1 after changeName1", s1.name)
    // john wick

    s1.changeName2("ethan hunt")
    fmt.Println("s1 after changeName2", s1.name)
    // ethan hunt
}
```

Output:

```
[novalagung:chapter-24 $ go run 2-method-pointer.go
s1 before john wick
---> on changeName1, name changed to jason bourne
s1 after changeName1 john wick
---> on changeName2, name changed to ethan hunt
s1 after changeName2 ethan hunt
```

Setelah eksekusi statement `s1.changeName1("jason bourne")`, nilai `s1.name` tidak berubah. Sebenarnya nilainya berubah tapi hanya dalam method `changeName1()` saja, nilai pada reference di objek-nya tidak berubah. Karena itulah ketika objek di print value dari `s1.name` tidak berubah.

Keistimewaan lain method pointer adalah, method itu sendiri bisa dipanggil dari objek pointer maupun objek biasa.

```
// pengaksesan method dari variabel objek biasa
var s1 = student{"john wick", 21}
s1.sayHello()

// pengaksesan method dari variabel objek pointer
var s2 = &student{"ethan hunt", 22}
s2.sayHello()
```

Berikut adalah penjelasan tambahan mengenai beberapa hal pada bab ini.

### A.24.3. Penggunaan Fungsi `strings.Split()`

Di bab ini ada fungsi baru yang kita gunakan: `strings.Split()`. Fungsi ini berguna untuk memisah string menggunakan pemisah yang ditentukan sendiri. Hasilnya adalah array berisikan kumpulan substring.

```
strings.Split("ethan hunt", " ")
// ["ethan", "hunt"]
```

Pada contoh di atas, string `"ethan hunt"` dipisah menggunakan separator spasi `" "`. Maka hasilnya terbentuk array berisikan 2 data, `"ethan"` dan `"hunt"`.

### A.24.4. Apakah `fmt.Println()` & `strings.Split()` Juga Merupakan Method ?

Setelah tahu apa itu method dan bagaimana penggunaannya, mungkin akan muncul di benak kita bahwa kode seperti `fmt.Println()`, `strings.Split()` dan lainnya-yang-berada-pada-package-lain juga merupakan method.

Tapi sayangnya **bukan**. `fmt` disitu bukanlah variabel objek, dan `Println()` bukan merupakan method-nya.

`fmt` adalah nama **package** yang di-import (bisa dilihat pada kode `import "fmt"`). Sedangkan `Println()` adalah **nama fungsi**. Untuk mengakses fungsi yang berada pada package lain, harus dituliskan nama package-nya. Hal ini berlaku juga di dalam package `main`. Jika ada fungsi dalam package `main` yang diakses dari package lain yang berbeda, maka penulisannya `main.NamaFungsi()`.

Lebih detailnya akan dibahas di bab selanjutnya.

## A.25. Property Public Dan Private

Bab ini membahas mengenai *modifier* public dan private dalam Golang. Kapan sebuah struct, fungsi, atau method bisa diakses dari package lain dan kapan tidak.

### A.25.0. PERINGATAN

Peringatan ini ditulis karena sudah terlalu banyak email yang penulis dapati, perihal error yang muncul ketika mengikuti pembahasan bab ini.

Bab ini memiliki beberapa perbedaan dibanding bab lainnya. Jika pembaca mengikuti secara berurutan, membaca penjelasan dan pembahasan yang sudah tertulis, **pasti akan mendapatkan 3 buah error**. Di tiap-tiap error, sebenarnya sudah terlampir:

1. Screenshot error
2. Penjelasan penyebab error
3. Cara resolve atau solusi dari ketiga error tersebut.

Kesimpulan dari email-email yang penulis dapati: **pembaca tidak tau cara mengatasi error tersebut, yang padahal sudah jelas tertulis akan muncul error dan juga cara mengatasinya. Ini kemungkinan besar disebabkan karena pembaca hanya copy-paste source code, tanpa membaca penjelasan-penjelasan yang padahal sudah tertulis cukup mendetail**. Tapi ini tidak semua, tapi banyak, jadi *no hard feeling*.

Oleh karena itu, JANGAN CUMA COPAS SOURCE KODE, BACA, PELAJARI, DAN PAHAM!

### A.25.1. Package Public & Private

Pengembangan aplikasi dalam *real development* pasti membutuhkan banyak sekali file program. Tidak mungkin dalam satu project semua file memiliki nama package `main`, biasanya akan dipisah sebagai package berbeda sesuai bagiannya.

Project folder selain berisikan file-file `.go` juga bisa berisikan folder. Di bahasa Golang, setiap satu folder atau subfolder adalah satu package, file-file yang ada didalamnya harus memiliki nama package yang berbeda dengan file di folder lain.

Dalam sebuah package, biasanya kita menulis sangat banyak komponen, entah itu fungsi, struct, variabel, atau lainnya. Komponen tersebut bisa leluasa digunakan dalam package yang sama. Contoh sederhananya seperti program yang telah kita praktekan di bab sebelum-sebelumnya, dalam package `main` ada banyak yang di-define: fungsi, variabel, closure, struct, dan lainnya; kesemuanya bisa langsung dimanfaatkan.

Jika dalam satu program terdapat lebih dari 1 package, atau ada package lain selain `main`, tidak bisa komponen dalam package lain tersebut diakses secara bebas dari `main`, karena tiap komponen memiliki hak akses.

Ada 2 jenis hak akses:

- Akses Public, menandakan komponen tersebut diperbolehkan untuk diakses dari package lain yang berbeda
- Akses Private, berarti komponen hanya bisa diakses dalam package yang sama, bisa dalam satu file saja atau dalam beberapa file yang masih 1 folder.

Penentuan hak akses yang tepat untuk tiap komponen sangatlah penting.

Di Golang cara menentukan level akses atau modifier sangat mudah, penandanya adalah **character case** karakter pertama nama fungsi, struct, variabel, atau lainnya. Ketika namanya diawali dengan huruf kapital menandakan bahwa modifier-nya public. Dan sebaliknya, jika diawali huruf kecil, berarti private.

## A.25.2. Penggunaan Package, Import, Dan Modifier Public & Private

Agar lebih mudah dipahami, maka langsung saja kita praktikan.

Pertama buat folder proyek baru bernama `belajar-golang-level-akses` di dalam folder `$GOPATH/src`, didalamnya buat file baru bernama `main.go`, set nama package file tersebut sebagai `main`.

Selanjutnya buat folder baru bernama `library` di dalam folder baru yang sudah dibuat tadi. Didalamnya buat file baru `library.go`, set nama package-nya `library`.

▼	go	Today, 5:07 PM	--
►	bin	Today, 5:07 PM	--
►	pkg	Today, 5:07 PM	--
▼	src	Today, 5:09 PM	--
►	belajar-golang-level-akses	Today, 5:11 PM	--
▼	library	Today, 5:11 PM	--
	library.go	Today, 5:02 PM	Zero bytes
	main.go	Today, 5:02 PM	Zero bytes

Buka file `library.go` lalu isi dengan kode berikut.

```
package library

import "fmt"

func SayHello() {
    fmt.Println("hello")
}

func introduce(name string) {
    fmt.Println("nama saya", name)
}
```

File `library.go` yang telah dibuat ditentukan nama package-nya adalah `library` (sesuai dengan nama folder), berisi dua buah fungsi, `SayHello()` dan `introduce()`.

- Fungsi `SayHello()`, level aksesnya adalah publik, ditandai dengan nama fungsi diawali huruf besar.
- Fungsi `introduce()` dengan level akses private, ditandai oleh huruf kecil di awal nama fungsi.

Selanjutnya kita lakukan tes apakah memang fungsi yang ber-modifier private dalam package `library` tidak bisa diakses dari package lain.

Buka file `main.go`, lalu tulis kode berikut.

```
package main

import "belajar-golang-level-akses/library"

func main() {
    library.SayHello()
    library.introduce("ethan")
}
```

Bisa dilihat bahwa package `library` yang telah dibuat tadi, di-import ke dalam package `main`.

Folder utama atau root folder dalam project yang sedang digarap adalah `belajar-golang-level-akses`, sehingga untuk import package lain yang merupakan subfolder, harus dituliskan lengkap path folder nya, seperti `"belajar-golang-level-akses/library"`.

Penanda root folder adalah, file di dalam folder tersebut package-nya `main`, dan file tersebut dijalankan menggunakan `go run`.

Perhatikan kode berikut.

```
library.SayHello()
library.introduce("ethan")
```

Cara pemanggilan fungsi yang berada dalam package lain adalah dengan menuliskan nama package target diikut dengan nama fungsi menggunakan *dot notation* atau tanda titik, seperti `library.SayHello()` atau `library.introduce("ethan")`

OK, sekarang coba jalankan kode yang sudah disiapkan di atas, hasilnya error.

```
[novalagung:belajar-golang-level-akses $ go run main.go
# command-line-arguments
./main.go:7: cannot refer to unexported name library.introduce
./main.go:7: undefined: library.introduce
novalagung:belajar-golang-level-akses $ ]
```

Error di atas disebabkan karena fungsi `introduce()` yang berada dalam package `library` memiliki level akses **private**, fungsi ini tidak bisa diakses dari package lain (pada kasus ini `main`). Agar bisa diakses, solusinya bisa dengan menjadikannya public, atau diubah cara pemanggilannya. Disini kita menggunakan cara ke-2.

Tambahkan parameter `name` pada fungsi `SayHello()`, lalu panggil fungsi `introduce()` dengan menyisipkan parameter `name` dari dalam fungsi `SayHello()`.

```
func SayHello(name string) {
    fmt.Println("hello")
    introduce(name)
}
```

Di `main`, cukup panggil fungsi `SayHello()` saja, sisipkan sebuah string sebagai parameter.

```
func main() {
    library.SayHello("ethan")
}
```

Coba jalankan lagi.

```
[novalagung:belajar-golang-level-akses $ go run main.go
hello
nama saya ethan
novalagung:belajar-golang-level-akses $ ]
```

## 25.3. Penggunaan Public & Private Pada Struct Dan Propertinya

Level akses private dan public juga bisa diterapkan di fungsi, struct, method, maupun property variabel. Cara penggunaanya sama seperti pada pembahasan sebelumnya, yaitu dengan menentukan **character case** huruf pertama nama komponen, apakah huruf besar atau kecil.

Belajar tentang level akses di Golang akan lebih cepat jika langsung praktik. Oleh karena itu langsung saja. Hapus isi file `library.go`, buat struct baru dengan nama `student` didalamnya.

```
package library

type student struct {
    Name string
    grade int
}
```

Buat contoh sederhana penerapan struct di atas pada file `main.go`.

```
package main

import "belajar-golang-level-akses/library"
import "fmt"

func main() {
    var s1 = library.student{"ethan", 21}
    fmt.Println("name ", s1.Name)
    fmt.Println("grade", s1.grade)
}
```

Setelah itu jalankan program.

```
[novalagung:belajar-golang-level-akses $ go run main.go
# command-line-arguments
./main.go:7: cannot refer to unexported name library.student
./main.go:7: undefined: library.student
novalagung:belajar-golang-level-akses $ ]
```

Error muncul lagi, kali ini penyebabnya adalah karena struct `student` masih di set sebagai private. Ganti menjadi public (dengan cara mengubah huruf awalnya menjadi huruf besar) lalu jalankan.

```
// pada library/library.go
type Student struct {
    Name string
    grade int
}

// pada main.go
var s1 = library.Student{"ethan", 21}
fmt.Println("name ", s1.Name)
fmt.Println("grade", s1.grade)
```

Output:

```
[novalagung:belajar-golang-level-akses $ go run main.go
# command-line-arguments
./main.go:7: implicit assignment of unexported field 'grade' in library.Student
literal
novalagung:belajar-golang-level-akses $ ]
```

Error masih tetap muncul, tapi kali ini berbeda. Error yang baru ini disebabkan karena salah satu properti dari struct `student` bermodifier private. Properti yg dimaksud adalah `grade`. Ubah menjadi public, lalu jalankan lagi.

```
// pada library/library.go
type Student struct {
    Name string
    Grade int
}

// pada main.go
var s1 = library.Student{"ethan", 21}
```

```
fmt.Println("name ", s1.Name)
fmt.Println("grade", s1.Grade)
```

Dari contoh program di atas, bisa disimpulkan bahwa untuk menggunakan `struct` yang berada di package lain, selain nama struct-nya harus bermodifier public, properti yang diakses juga harus public.

```
[novalagung:belajar-golang-level-akses $ go run main.go
 name ethan
 grade 21
novalagung:belajar-golang-level-akses $ ]
```

## 25.4. Import Dengan Prefix Tanda Titik

Seperti yang kita tahu, untuk mengakses fungsi/struct/variabel yg berada di package lain, nama package nya perlu ditulis, contohnya seperti pada penggunaan penggunaan `library.Student` dan `fmt.Println()`.

Di Golang, komponen yang berada di package lain yang di-import bisa dijadikan se-level dengan komponen package peng-import, caranya dengan menambahkan tanda titik ( . ) setelah penulisan keyword `import`. Maksud dari se-level disini adalah, semua properti di package lain yg di-import bisa diakses tanpa perlu menuliskan nama package, seperti ketika mengakses sesuatu dari file yang sama.

```
import (
    . "belajar-golang-level-akses/library"
    "fmt"
)

func main() {
    var s1 = Student{"ethan", 21}
    fmt.Println("name ", s1.Name)
    fmt.Println("grade", s1.Grade)
}
```

Pada kode di atas package `library` di-import menggunakan tanda titik. Dengan itu, pemanggilan struct `Student` tidak perlu dengan menuliskan nama package nya.

## 25.5. Pemanfaatan Alias Ketika Import Package

Fungsi yang berada di package lain bisa diakses dengan cara menuliskan nama-package diikuti nama fungsi-nya, contohnya seperti `fmt.Println()`. Package yang sudah di-import tersebut bisa diubah namanya dengan cara menggunakan alias pada saat import. Contohnya bisa dilihat pada kode berikut.

```
import (
    f "fmt"
)

func main() {
    f.Println("Hello World!")
}
```

Pada kode di-atas, package `fmt` di tentukan aliasnya adalah `f`, untuk mengakses `Println()` cukup dengan `f.Println()`.

## 25.6. Mengakses Properti Dalam File Yang Package-nya Sama

Jika properti yang ingin di akses masih dalam satu package tapi berbeda file, cara mengaksesnya bisa langsung dengan memanggil namanya. Hanya saja ketika eksekusi, file-file lain yang nama package-nya sama juga ikut dipanggil.

Langsung saja kita praktikan, buat file baru dalam `belajar-golang-level-akses` dengan nama `partial.go`.

▼ go		Today, 5:07 PM	--
► bin		Today, 5:07 PM	--
► pkg		Today, 5:07 PM	--
▼ src		Today, 5:09 PM	--
▼ belajar-golang-level-akses		Today, 5:46 PM	--
► library		Today, 5:11 PM	--
└ main.go		Today, 5:02 PM	Zero bytes
└ partial.go		Today, 5:02 PM	Zero bytes

Tulis kode berikut pada file `partial.go`. File ini kita set package-nya `main` (sama dengan nama package file `main.go`).

```
package main

import "fmt"

func sayHello(name string) {
    fmt.Println("halo", name)
}
```

Hapus semua isi file `main.go`, lalu silakan tulis kode berikut.

```
package main

func main() {
    sayHello("ethan")
}
```

Sekarang terdapat 2 file berbeda (`main.go` dan `partial.go`) dengan package adalah sama, `main`. Pada saat `go build` atau `go run`, semua file dengan nama package `main` harus dituliskan sebagai argumen command.

```
$ go run main.go partial.go
```

Fungsi `sayHello` pada file `partial.go` bisa dikenali meski level aksesnya adalah private. Hal ini karena kedua file tersebut (`main.go` dan `partial.go`) memiliki package yang sama.

Cara lain untuk menjalankan program bisa dengan perintah `go run *.go`, dengan cara ini tidak perlu menuliskan nama file nya satu per satu.

```
[novalagung:belajar-golang-level-akses $ go run main.go partial.go
halo ethan
novalagung:belajar-golang-level-akses $ ]
```

## 25.7. Fungsi `init()`

Selain fungsi `main()`, terdapat juga fungsi spesial, yaitu `init()`. Fungsi ini otomatis dipanggil pertama kali ketika aplikasi di-run. Jika fungsi ini berada dalam package `main`, maka dipanggil lebih dulu sebelum fungsi `main()` dieksekusi.

Langsung saja kita praktikkan. Buka file `library.go`, hapus isinya lalu isi dengan kode berikut.

```

package library

import "fmt"

var Student = struct {
    Name string
    Grade int
} {}

func init() {
    Student.Name = "John Wick"
    Student.Grade = 2

    fmt.Println("--> library/library.go imported")
}

```

Pada package tersebut, variabel `Student` dibuat dengan isi anonymous struct. Dalam fungsi `init`, nilai `Name` dan `Grade` variabel di-set.

Selanjutnya buka file `main.go`, isi dengan kode berikut.

```

package main

import "belajar-golang-level-akses/library"
import "fmt"

func main() {
    fmt.Printf("Name : %s\n", library.Student.Name)
    fmt.Printf("Grade : %d\n", library.Student.Grade)
}

```

Package `library` di-import, dan variabel `Student` dikonsumsi. Pada saat import package, fungsi `init()` yang berada didalamnya langsung dieksekusi.

Property variabel objek `Student` akan diisi dan sebuah pesan ditampilkan ke console.

Dalam sebuah package diperbolehkan ada banyak fungsi `init()` (urutan eksekusinya adalah sesuai file mana yg terlebih dahulu digunakan). Fungsi ini dipanggil sebelum fungsi `main()`, pada saat eksekusi program.

```

novalagung:belajar-golang-import-init $ go run main.go
--> library/library.go imported
Name : John Wick
Grade : 2
novalagung:belajar-golang-import-init $

```

## A.26. Interface

Interface adalah kumpulan definisi method yang tidak memiliki isi (hanya definisi saja), dan dibungkus dengan nama tertentu.

Interface merupakan tipe data. Nilai objek bertipe interface default-nya adalah `nil`. Interface mulai bisa digunakan jika sudah ada isinya, yaitu objek konkret yang memiliki definisi method minimal sama dengan yang ada di interface-nya.

### A.26.1. Penerapan Interface

Yang pertama perlu dilakukan untuk menerapkan interface adalah menyiapkan interface beserta definisi method nya. Keyword `type` dan `interface` digunakan untuk pendefinisian interface.

```
package main

import "fmt"
import "math"

type hitung interface {
    luas() float64
    keliling() float64
}
```

Pada kode di atas, interface `hitung` memiliki 2 definisi method, `luas()` dan `keliling()`. Interface ini nantinya digunakan sebagai tipe data pada variabel, dimana variabel tersebut akan menampung menampung objek bangun datar hasil dari struct yang akan kita buat.

Dengan memanfaatkan interface `hitung`, perhitungan luas dan keliling bangun datar bisa dilakukan, tanpa perlu tahu jenis bangun datarnya sendiri itu apa.

Siapkan struct bangun datar `lingkaran`, struct ini memiliki method yang beberapa diantaranya terdefinisi di interface `hitung`.

```
type lingkaran struct {
    diameter float64
}

func (l lingkaran) jariJari() float64 {
    return l.diameter / 2
}

func (l lingkaran) luas() float64 {
    return math.Pi * math.Pow(l.jariJari(), 2)
}

func (l lingkaran) keliling() float64 {
    return math.Pi * l.diameter
}
```

Struct `lingkaran` di atas memiliki tiga method, `jariJari()`, `luas()`, dan `keliling()`.

Selanjutnya, siapkan struct bangun datar `persegi`.

```
type persegi struct {
    sisi float64
```

```

    }

func (p persegi) luas() float64 {
    return math.Pow(p.sisi, 2)
}

func (p persegi) keliling() float64 {
    return p.sisi * 4
}

```

Perbedaan struct `persegi` dengan `lingkaran` terletak pada method `jariJari()`. Struct `persegi` tidak memiliki method tersebut. Tetapi meski demikian, variabel objek hasil cetakan 2 struct ini akan tetap bisa ditampung oleh variabel cetakan interface `hitung`, karena dua method yang ter-definisi di interface tersebut juga ada pada struct `persegi` dan `lingkaran`, yaitu `luas()` dan `keliling()`.

Buat implementasi perhitungan di `main`.

```

func main() {
    var bangunDatar hitung

    bangunDatar = persegi{10.0}
    fmt.Println("===== persegi")
    fmt.Println("luas      :", bangunDatar.luas())
    fmt.Println("keliling   :", bangunDatar.keliling())

    bangunDatar = lingkaran{14.0}
    fmt.Println("===== lingkaran")
    fmt.Println("luas      :", bangunDatar.luas())
    fmt.Println("keliling   :", bangunDatar.keliling())
    fmt.Println("jari-jari  :", bangunDatar.(lingkaran).jariJari())
}

```

Perhatikan kode di atas. Variabel objek `bangunDatar` bertipe interface `hitung`. Variabel tersebut digunakan untuk menampung objek konkret buatan struct `lingkaran` dan `persegi`.

Dari variabel tersebut, method `luas()` dan `keliling()` diakses. Secara otomatis Golang akan mengarahkan pemanggilan method pada interface ke method asli milik struct yang bersangkutan.

```
[novalagung:belajar-golang $ go run bab26.go
===== persegi
luas      : 100
keliling   : 40
===== lingkaran
luas      : 153.93804002589985
keliling   : 43.982297150257104
jari-jari  :
novalagung:belajar-golang $ ]
```

Method `jariJari()` pada struct `lingkaran` tidak akan bisa diakses karena tidak terdefinisi dalam interface `hitung`. Pengaksesannya dengan paksa akan menyebabkan error.

Untuk mengakses method yang tidak ter-definisi di interface, variabel-nya harus di-casting terlebih dahulu ke tipe asli variabel konkretnya (pada kasus ini tipenya `lingkaran`), setelahnya method akan bisa diakses.

Cara casting objek interface sedikit unik, yaitu dengan menuliskan nama tipe tujuan dalam kurung, ditempatkan setelah nama interface dengan menggunakan notasi titik (seperti cara mengakses property, hanya saja ada tanda kurung nya). Contohnya bisa dilihat di kode berikut. Statement `bangunDatar.(lingkaran)` adalah contoh casting pada objek interface.

```

var bangunDatar hitung = lingkaran{14.0}
var bangunLingkaran lingkaran = bangunDatar.(lingkaran)

bangunLingkaran.jariJari()

```

Perlu diketahui juga, jika ada interface yang menampung objek konkret dimana struct-nya tidak memiliki salah satu method yang terdefinisi di interface, error juga akan muncul. Intinya kembali ke aturan awal, variabel interface hanya bisa menampung objek yang minimal memiliki semua method yang terdefinisi di interface-nya.

## A.26.2. Embedded Interface

Interface bisa di-embed ke interface lain, sama seperti struct. Cara penerapannya juga sama, cukup dengan menuliskan nama interface yang ingin di-embed ke dalam interface tujuan.

Pada contoh berikut, disiapkan interface bernama `hitung2d` dan `hitung3d`. Kedua interface tersebut kemudian di-embed ke interface baru bernama `hitung`.

```
package main

import "fmt"
import "math"

type hitung2d interface {
    luas() float64
    keliling() float64
}

type hitung3d interface {
    volume() float64
}

type hitung interface {
    hitung2d
    hitung3d
}
```

Interface `hitung2d` berisikan method untuk kalkulasi luas dan keliling, sedang `hitung3d` berisikan method untuk mencari volume bidang. Kedua interface tersebut diturunkan di interface `hitung`, menjadikannya memiliki kemampuan untuk menghitung luas, keliling, dan volume.

Next, siapkan struct baru bernama `kubus` yang memiliki method `luas()`, `keliling()`, dan `volume()`.

```
type kubus struct {
    sisi float64
}

func (k *kubus) volume() float64 {
    return math.Pow(k.sisi, 3)
}

func (k *kubus) luas() float64 {
    return math.Pow(k.sisi, 2) * 6
}

func (k *kubus) keliling() float64 {
    return k.sisi * 12
}
```

Objek hasil cetakan struct `kubus` di atas, nantinya akan ditampung oleh objek cetakan interface `hitung` yang isinya merupakan gabungan interface `hitung2d` dan `hitung3d`.

Terakhhir, buat implementasi-nya di main.

```
func main() {
    var bangunRuang hitung = &kubus{4}
```

```
fmt.Println("===== kubus")
fmt.Println("luas      :", bangunRuang.luas())
fmt.Println("keliling  :", bangunRuang.keliling())
fmt.Println("volume    :", bangunRuang.volume())
}
```

Bisa dilihat di kode di atas, lewat interface `hitung`, method `luas`, `keliling`, dan `volume` bisa di akses.

Pada bab 24 dijelaskan bahwa method pointer bisa diakses lewat variabel objek biasa dan variabel objek pointer. Variabel objek yang dicetak menggunakan struct yang memiliki method pointer, jika ditampung kedalam variabel interface, harus diambil referensi-nya terlebih dahulu. Contohnya bisa dilihat pada kode di atas `var bangunRuang hitung = &kubus{4} .`

```
[novalagung:belajar-golang $ go run bab26.go
=====
kubus
luas      : 96
keliling  : 48
volume    : 64
novalagung:belajar-golang $ ]
```

## A.27. Interface Kosong

Interface kosong atau `interface{}` adalah tipe data yang sangat spesial. Variabel bertipe ini bisa menampung segala jenis data, bahkan array, bisa pointer bisa tidak (konsep ini disebut dengan **dynamic typing**).

### A.27.1. Penggunaan `interface{}`

`interface{}` merupakan tipe data, sehingga cara penggunaannya sama seperti pada tipe data lainnya, hanya saja nilai yang diisikan bisa apa saja. Contoh:

```
package main

import "fmt"

func main() {
    var secret interface{}

    secret = "ethan hunt"
    fmt.Println(secret)

    secret = []string{"apple", "manggo", "banana"}
    fmt.Println(secret)

    secret = 12.4
    fmt.Println(secret)
}
```

Keyword `interface` seperti yang kita tau, digunakan untuk pembuatan interface. Tetapi ketika ditambahkan kurung kurawal (`{}`) di belakangnya (menjadi `interface{}`), maka kegunaannya akan berubah, yaitu sebagai tipe data.

```
[novalagung:belajar-golang $ go run bab27.go
data 1: ethan hunt
data 2: [apple manggo banana]
data 3: 12.4
novalagung:belajar-golang $ ]
```

Agar tidak bingung, coba perhatikan kode berikut.

```
var data map[string]interface{}

data = map[string]interface{}{
    "name":      "ethan hunt",
    "grade":     2,
    "breakfast": []string{"apple", "manggo", "banana"},
}
```

Pada kode di atas, disiapkan variabel `data` dengan tipe `map[string]interface{}`, yaitu sebuah koleksi dengan key bertipe `string` dan nilai bertipe interface kosong `interface{}`.

Kemudian variabel tersebut di-inisialisasi, ditambahkan lagi kurung kurawal setelah keyword deklarasi untuk kebutuhan pengisian data, `map[string]interface{}{ /* data */ }`.

Dari situ terlihat bahwa `interface{}` bukanlah sebuah objek, melainkan tipe data.

### A.27.2. Casting Variabel Interface Kosong

Variabel bertipe `interface{}` bisa ditampilkan ke layar sebagai `string` dengan memanfaatkan fungsi `print`, seperti `fmt.Println()`. Tapi perlu diketahui bahwa nilai yang dimunculkan tersebut bukanlah nilai asli, melainkan bentuk `string` dari nilai aslinya.

Hal ini penting diketahui, karena untuk melakukan operasi yang membutuhkan nilai asli pada variabel yang bertipe `interface{}`, diperlukan casting ke tipe aslinya. Contoh seperti pada kode berikut.

```
package main

import "fmt"
import "strings"

func main() {
    var secret interface{}

    secret = 2
    var number = secret.(int) * 10
    fmt.Println(secret, "multiplied by 10 is :", number)

    secret = []string{"apple", "manggo", "banana"}
    var gruits = strings.Join(secret.([]string), ", ")
    fmt.Println(gruits, "is my favorite fruits")
}
```

Pertama, variabel `secret` menampung nilai bertipe numerik. Ada kebutuhan untuk mengalikan nilai yang ditampung variabel tersebut dengan angka `10`. Maka perlu dilakukan casting ke tipe aslinya, yaitu `int`, setelahnya barulah nilai bisa dioperasikan, yaitu `secret.(int) * 10`.

Pada contoh kedua, `secret` berisikan array string. Kita memerlukan string tersebut untuk digabungkan dengan pemisah tanda koma. Maka perlu di-casting ke `[]string` terlebih dahulu sebelum bisa digunakan di `strings.Join()`, contohnya pada `strings.Join(secret.([]string), ", ")`.

```
[novalagung:belajar-golang $ go run bab27.go
2 multiplied by 10 is : 20
apple, manggo, banana is my favorite fruits
novalagung:belajar-golang $ ]
```

Teknik casting pada interface disebut dengan **type assertions**.

### A.27.3. Casting Variabel Interface Kosong Ke Objek Pointer

Variabel `interface{}` bisa menyimpan data apa saja, termasuk data objek, pointer, ataupun gabungan keduanya. Di bawah ini merupakan contoh penerapan interface untuk menampung data objek pointer.

```
type person struct {
    name string
    age int
}

var secret interface{} = &person{name: "wick", age: 27}
var name = secret.(*person).name
fmt.Println(name)
```

Variabel `secret` dideklarasikan bertipe `interface{}` menampung referensi objek cetakan struct `person`. Cara casting dari `interface{}` ke struct pointer adalah dengan menuliskan nama struct-nya dan ditambahkan tanda asterisk (`*`) di awal, contohnya seperti `secret.(*person)`. Setelah itu barulah nilai asli bisa diakses.

```
[novalagung:belajar-golang $ go run bab27.go
wick
novalagung:belajar-golang $ ]
```

## A.27.4. Kombinasi Slice, map , dan interface{}

Tipe `[]map[string]interface{}` adalah salah satu tipe yang paling sering digunakan (menurut saya), karena tipe data tersebut bisa menjadi alternatif tipe slice struct.

Pada contoh berikut, variabel `person` dideklarasikan berisi data slice `map` berisikan 2 item dengan key adalah `name` dan `age`.

```
var person = []map[string]interface{}{
    {"name": "Wick", "age": 23},
    {"name": "Ethan", "age": 23},
    {"name": "Bourne", "age": 22},
}

for _, each := range person {
    fmt.Println(each["name"], "age is", each["age"])
}
```

Dengan memanfaatkan slice dan `interface{}`, kita bisa membuat data array yang isinya adalah bisa apa saja. Silakan perhatikan contoh berikut.

```
var fruits = []interface{}{
    map[string]interface{}{"name": "strawberry", "total": 10},
    []string{"manggo", "pineapple", "papaya"},
    "orange",
}

for _, each := range fruits {
    fmt.Println(each)
}
```

## A.28. Reflect

Reflection adalah teknik untuk inspeksi variabel, mengambil informasi dari variabel tersebut atau bahkan memanipulasinya. Cakupan informasi yang bisa didapatkan lewat reflection sangat luas, seperti melihat struktur variabel, tipe, nilai pointer, dan banyak lagi.

Golang menyediakan package bernama `reflect`, berisikan banyak sekali fungsi untuk keperluan reflection. Di bab ini, kita akan belajar tentang dasar penggunaan package tersebut.

Dari banyak fungsi yang tersedia di dalam package tersebut, ada 2 fungsi yang paling penting untuk diketahui, yaitu `reflect.ValueOf()` dan `reflect.TypeOf()`.

- Fungsi `reflect.ValueOf()` akan mengembalikan objek dalam tipe `reflect.Value`, yang berisikan informasi yang berhubungan dengan nilai pada variabel yang dicari
- Sedangkan `reflect.TypeOf()` mengembalikan objek dalam tipe `reflect.Type`. Objek tersebut berisikan informasi yang berhubungan dengan tipe data variabel yang dicari

### A.28.1. Mencari Tipe Data & Value Menggunakan Reflect

Dengan reflection, tipe data dan nilai variabel dapat diketahui dengan mudah. Contoh penerapannya bisa dilihat pada kode berikut.

```
package main

import "fmt"
import "reflect"

func main() {
    var number = 23
    var reflectValue = reflect.ValueOf(number)

    fmt.Println("tipe variabel :", reflectValue.Type())

    if reflectValue.Kind() == reflect.Int {
        fmt.Println("nilai variabel :", reflectValue.Int())
    }
}
```

```
[novalagung:belajar-golang $ go run bab28.go
tipe variabel : int
nilai variabel : 23
novalagung:belajar-golang $ ]
```

Fungsi `reflect.valueOf()` memiliki parameter yang bisa menampung segala jenis tipe data. Fungsi tersebut mengembalikan objek dalam tipe `reflect.Value`, yang berisikan informasi mengenai variabel yang bersangkutan.

Objek `reflect.Value` memiliki beberapa method, salah satunya `Type()`. Method ini mengembalikan tipe data variabel yang bersangkutan dalam bentuk `string`.

Statement `reflectValue.Int()` menghasilkan nilai `int` dari variabel `number`. Untuk menampilkan nilai variabel `reflect`, harus dipastikan dulu tipe datanya. Ketika tipe data adalah `int`, maka bisa menggunakan method `Int()`. Ada banyak lagi method milik struct `reflect.Value` yang bisa digunakan untuk pengambilan nilai dalam bentuk tertentu, contohnya: `reflectValue.String()` digunakan untuk mengambil nilai `string`, `reflectValue.Float64()` untuk nilai `float64`, dan lainnya.

Perlu diketahui, fungsi yang digunakan harus sesuai dengan tipe data nilai yang ditampung variabel. Jika fungsi yang digunakan berbeda dengan tipe data variabelnya, maka akan menghasilkan error. Contohnya pada variabel menampung nilai bertipe `float64`, maka tidak bisa menggunakan method `String()`.

Diperlukan adanya pengecekan tipe data nilai yang disimpan, agar pengambilan nilai bisa tepat. Salah satunya bisa dengan cara seperti kode di atas, yaitu dengan mengecek dahulu apa jenis tipe datanya menggunakan method `Kind()`, setelah itu diambil nilainya dengan method yang sesuai.

Berikut adalah konstanta tipe data dan method yang bisa digunakan dalam refleksi di GoLang.

- Bool
- Int
- Int8
- Int16
- Int32
- Int64
- Uint
- Uint8
- Uint16
- Uint32
- Uint64
- Uintptr
- Float32
- Float64
- Complex64
- Complex128
- Array
- Chan
- Func
- Interface
- Map
- Ptr
- Slice
- String
- Struct
- UnsafePointer

## Pengaksesan Nilai Dalam Bentuk `interface{}`

Jika nilai hanya diperlukan untuk ditampilkan ke output, bisa menggunakan `.Interface()`. Lewat method tersebut segala jenis nilai bisa diakses dengan mudah.

```
var number = 23
var reflectValue = reflect.ValueOf(number)

fmt.Println("tipe variabel :", reflectValue.Type())
fmt.Println("nilai variabel :", reflectValue.Interface())
```

Fungsi `Interface()` mengembalikan nilai interface kosong atau `interface{}`. Nilai aslinya sendiri bisa diakses dengan meng-casting interface kosong tersebut.

```
var nilai = reflectValue.Interface().(int)
```

## A.28.2. Pengaksesan Informasi Property Variabel Objek

Reflect bisa digunakan untuk mengambil informasi semua property variabel objek cetakan struct, dengan catatan property-property tersebut bermodifier public. Langsung saja kita praktikan, siapkan sebuah struct bernama `student`.

```
type student struct {
    Name string
    Grade int
}
```

Buat method baru untuk struct tersebut, dengan nama method `getPropertyInfo()`. Method ini berisikan kode untuk mengambil dan menampilkan informasi tiap property milik struct `student`.

```
func (s *student) getPropertyInfo() {
    var reflectValue = reflect.ValueOf(s)

    if reflectValue.Kind() == reflect.Ptr {
        reflectValue = reflectValue.Elem()
    }

    var reflectType = reflectValue.Type()

    for i := 0; i < reflectValue.NumField(); i++ {
        fmt.Println("nama      :", reflectType.Field(i).Name)
        fmt.Println("tipe data :", reflectType.Field(i).Type)
        fmt.Println("nilai     :", reflectValue.Field(i).Interface())
        fmt.Println("")
    }
}
```

Terakhir, lakukan uji coba method di fungsi `main`.

```
func main() {
    var s1 = &student{Name: "wick", Grade: 2}
    s1.getPropertyInfo()
}
```

```
[novalagung:belajar-golang $ go run bab28.go
    nama      : Name
    tipe data : string
    nilai     : wick

    nama      : Grade
    tipe data : int
    nilai     : 2]
```

Didalam method `getPropertyInfo` terjadi beberapa hal. Pertama objek `reflect.Value` dari variabel `s` diambil. Setelah itu dilakukan pengecekan apakah variabel objek tersebut merupakan pointer atau tidak (bisa dilihat dari `if reflectValue.Kind() == reflect.Ptr`, jika bernilai `true` maka variabel adalah pointer). jika ternyata variabel memang pointer, maka perlu diambil objek reflect aslinya dengan cara memanggil method `Elem()`.

Setelah itu, dilakukan perulangan sebanyak jumlah property yang ada pada struct `student`. Method `NumField()` akan mengembalikan jumlah property publik yang ada dalam struct.

Di tiap perulangan, informasi tiap property struct diambil berurutan dengan lewat method `Field()`. Method ini ada pada tipe `reflect.Value` dan `reflect.Type`.

- `reflectType.Field(i).Name` akan mengembalikan nama property
- `reflectType.Field(i).Type` mengembalikan tipe data property

- `reflectValue.Field(i).Interface()` mengembalikan nilai property dalam bentuk `interface{}`

Pengambilan informasi property, selain menggunakan indeks, bisa diambil berdasarkan nama field dengan menggunakan method `FieldByName()`.

### A.28.3. Pengaksesan Informasi Method Variabel Objek

Informasi mengenai method juga bisa diakses lewat reflect, syaratnya masih sama seperti pada pengaksesan property, yaitu harus bermodifier public.

Pada contoh dibawah ini informasi method `SetName` akan diambil lewat reflection. Siapkan method baru di struct `student`, dengan nama `SetName`.

```
func (s *student) SetName(name string) {
    s.Name = name
}
```

Buat contoh penerapannya di fungsi `main`.

```
func main() {
    var s1 = &student{Name: "john wick", Grade: 2}
    fmt.Println("nama :", s1.Name)

    var reflectValue = reflect.ValueOf(s1)
    var method = reflectValue.MethodByName("SetName")
    method.Call([]reflect.Value{
        reflect.ValueOf("wick"),
    })

    fmt.Println("nama :", s1.Name)
}
```

```
[novalagung:belajar-golang $ go run bab28.go
nama : john wick
nama : wick
novalagung:belajar-golang $ ]
```

Pada kode di atas, disiapkan variabel `s1` yang merupakan instance struct `student`. Awalnya property `Name` variabel tersebut berisikan string `"john wick"`.

Setelah itu, refleksi nilai objek tersebut diambil, refleksi method `SetName` juga diambil. Pengambilan refleksi method dilakukan menggunakan `MethodByName` dengan argument adalah nama method yang diinginkan, atau bisa juga lewat indeks method-nya (menggunakan `Method(i)`).

Setelah refleksi method yang dicari sudah didapatkan, `call()` dipanggil untuk eksekusi method.

Jika eksekusi method diikuti pengisian parameter, maka parameternya harus ditulis dalam bentuk array `[]reflect.Value` berurutan sesuai urutan deklarasi parameter-nya. Dan nilai yang dimasukkan ke array tersebut harus dalam bentuk `reflect.Value` (gunakan `reflect.ValueOf()` untuk pengambilannya).

```
[]reflect.Value{
    reflect.ValueOf("wick"),
}
```



## A.29. Goroutine

Goroutine mirip dengan thread thread, tapi sebenarnya bukan. Sebuah *native thread* bisa berisikan sangat banyak goroutine. Mungkin lebih pas kalau goroutine disebut sebagai **mini thread**. Goroutine sangat ringan, hanya dibutuhkan sekitar **2kB** memori saja untuk satu buah goroutine. Eksepsi goroutine bersifat *asynchronous*, menjadikannya tidak saling tunggu dengan goroutine lain.

Karena goroutine sangat ringan, maka eksekusi banyak goroutine bukan masalah. Akan tetapi jika jumlah goroutine sangat banyak sekali (contoh 1 juta goroutine dijalankan pada komputer dengan RAM terbatas), memang proses akan jauh lebih cepat selesai, tapi memory / RAM juga bisa bengkak.

Goroutine merupakan salah satu bagian paling penting dalam Concurrent Programming di Golang. Salah satu yang membuat goroutine sangat istimewa adalah eksekusi-nya dijalankan di multi core processor. Kita bisa tentukan berapa banyak core yang aktif, makin banyak akan makin cepat.

Mulai bab 29 ini hingga bab 34, lalu dilanjut bab 56 dan 57, kita akan membahas tentang fitur-fitur yang disediakan golang untuk kebutuhan Concurrent Programming.

Concurrency atau konkurensi berbeda dengan paralel. Paralel adalah eksekusi banyak proses secara bersamaan. Sedangkan konkurensi adalah komposisi dari sebuah proses. Konkurensi merupakan struktur, sedangkan paralel adalah bagaimana eksekusinya berlangsung.

### A.29.1. Penerapan Goroutine

Untuk menerapkan goroutine, proses yang akan dieksekusi sebagai goroutine harus dibungkus kedalam sebuah fungsi. Pada saat pemanggilan fungsi tersebut, ditambahkan keyword `go` didepannya, dengan itu goroutine baru akan dibuat dengan tugas adalah menjalankan proses yang ada dalam fungsi tersebut.

Berikut merupakan contoh implementasi sederhana tentang goroutine. Program di bawah ini menampilkan 10 baris teks, 5 dieksekusi dengan cara biasa, dan 5 lainnya dieksekusi sebagai goroutine baru.

```
package main

import "fmt"
import "runtime"

func print(till int, message string) {
    for i := 0; i < till; i++ {
        fmt.Println((i + 1), message)
    }
}

func main() {
    runtime.GOMAXPROCS(2)

    go print(5, "halo")
    print(5, "apa kabar")

    var input string
    fmt.Scanln(&input)
}
```

Pada kode di atas, Fungsi `runtime.GOMAXPROCS(n)` digunakan untuk menentukan jumlah core yang diaktifkan untuk eksekusi program.

Pembuatan goroutine baru ditandai dengan keyword `go`. Contohnya pada statement `go print(5, "halo")`, di situ fungsi `print()` dieksekusi sebagai goroutine baru.

Fungsi `fmt.Scanln()` mengakibatkan proses jalannya aplikasi berhenti di baris itu (**blocking**) hingga user menekan tombol enter. Hal ini perlu dilakukan karena ada kemungkinan waktu selesainya eksekusi goroutine `print()` lebih lama dibanding waktu selesainya goroutine utama `main()`, mengingat bahwa keduanya sama-sama asynchronous. Jika itu terjadi, goroutine yang belum selesai secara paksa dihentikan prosesnya karena goroutine utama sudah selesai dijalankan.

```
[novalagung:belajar-golang $ go run bab29.go
1 apa kabar
2 apa kabar
3 apa kabar
1 halo
4 apa kabar
2 halo
5 apa kabar
3 halo
4 halo
5 halo

[novalagung:belajar-golang $ go run bab29.go
1 apa kabar
1 halo
2 apa kabar
3 apa kabar
4 apa kabar
5 apa kabar
2 halo
3 halo
4 halo
5 halo]
```

Bisa dilihat di output, tulisan `"halo"` dan `"apa kabar"` bermunculan selang-seling. Ini disebabkan karena statement `print(5, "halo")` dijalankan sebagai goroutine baru, menjadikannya tidak saling tunggu dengan `print(5, "apa kabar")`.

Pada gambar di atas, program dieksekusi 2 kali. Hasil eksekusi pertama berbeda dengan kedua, penyebabnya adalah karena kita menggunakan 2 prosesor. Goroutine mana yang dieksekusi terlebih dahulu tergantung kedua prosesor tersebut.

---

Berikut adalah penjelasan tambahan tentang beberapa fungsi yang baru kita pelajari di atas.

## A.29.2. Penggunaan Fungsi `runtime.GOMAXPROCS()`

Fungsi ini digunakan untuk menentukan jumlah core atau processor yang digunakan dalam eksekusi program.

Jumlah yang diinputkan secara otomatis akan disesuaikan dengan jumlah asli *logical processor* yang ada. Jika jumlahnya lebih, maka dianggap menggunakan sejumlah prosesor yang ada.

## A.29.3. Penggunaan Fungsi `fmt.Scanln()`

Fungsi ini akan meng-capture semua karakter sebelum user menekan tombol enter, lalu menyimpannya pada variabel.

```
func Scanln(a ...interface{}) (n int, err error)
```

Kode di atas merupakan skema fungsi `fmt.Scanln()`. Fungsi tersebut bisa menampung parameter bertipe `interface{}` berjumlah tak terbatas. Tiap parameter akan menampung karakter-karakter inputan user yang sudah dipisah dengan tanda spasi. Agar lebih jelas, silakan perhatikan contoh berikut.

```
var s1, s2, s3 string
fmt.Scanln(&s1, &s2, &s3)

// user inputs: "trafalgar d law"

fmt.Println(s1) // trafalgar
fmt.Println(s2) // d
fmt.Println(s3) // law
```

Bisa dilihat pada kode di atas, untuk menampung inputan text `trafalgar d law`, dibutuhkan 3 buah variabel. Juga perlu diperhatikan bahwa yang disisipkan sebagai parameter pada pemanggilan fungsi `fmt.Scanln()` adalah referensi variabel, bukan nilai aslinya.

## A.30. Channel

**Channel** digunakan untuk menghubungkan goroutine satu dengan goroutine lain. Mekanismenya adalah serah-terima data lewat channel tersebut. Goroutine pengirim dan penerima harus berada pada channel yang berbeda (konsep ini disebut **buffered channel**). Pengiriman dan penerimaan data pada channel bersifat **blocking** atau **synchronous**.

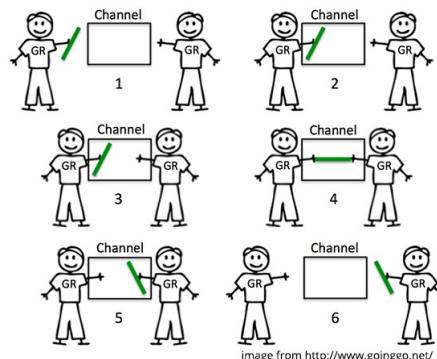


Image from <http://www.goinggo.net/>

Pada bab ini kita akan belajar mengenai pemanfaatan channel.

### A.30.1. Penerapan Channel

Channel merupakan sebuah variabel, dibuat dengan menggunakan keyword `make` dan `chan`. Variabel channel memiliki tugas menjadi pengirim dan penerima data.

Program berikut adalah contoh implementasi channel. 3 buah goroutine dieksekusi, di masing-masing goroutine terdapat proses pengiriman data lewat channel. Data tersebut akan diterima 3 kali di goroutine utama `main`.

```
package main

import "fmt"
import "runtime"

func main() {
    runtime.GOMAXPROCS(2)

    var messages = make(chan string)

    var sayHelloTo = func(who string) {
        var data = fmt.Sprintf("hello %s", who)
        messages <- data
    }

    go sayHelloTo("john wick")
    go sayHelloTo("ethan hunt")
    go sayHelloTo("jason bourne")

    var message1 = <-messages
    fmt.Println(message1)

    var message2 = <-messages
    fmt.Println(message2)

    var message3 = <-messages
    fmt.Println(message3)
}
```

Pada kode di atas, variabel `messages` dideklarasikan bertipe `channel string`. Cara pembuatan channel yaitu dengan menuliskan keyword `make` dengan isi keyword `chan` diikuti dengan tipe data channel yang diinginkan.

```
var messages = make(chan string)
```

Selain itu disiapkan juga closure `sayHelloTo` yang menghasilkan data string. Data tersebut dikirim lewat channel `messages`. Tanda `<-` jika dituliskan di sebelah kiri nama variabel, berarti sedang berlangsung proses pengiriman data dari variabel yang berada di kanan lewat channel yang berada di kiri (pada konteks ini, variabel `data` dikirim lewat channel `messages`).

```
var sayHelloTo = func(who string) {
    var data = fmt.Sprintf("hello %s", who)
    messages <- data
}
```

Fungsi `sayHelloTo` dieksekusi tiga kali sebagai goroutine berbeda. Menjadikan tiga proses ini berjalan secara **asynchronous** atau tidak saling tunggu.

```
go sayHelloTo("john wick")
go sayHelloTo("ethan hunt")
go sayHelloTo("jason bourne")
```

Dari ketiga fungsi tersebut, goroutine yang selesai paling awal akan mengirim data lebih dulu, datanya kemudian diterima variabel `message1`. Tanda `<-` jika dituliskan di sebelah kanan channel, menandakan proses penerimaan data dari channel yang di kanan, untuk disimpan ke variabel yang di kiri.

```
var message1 = <-messages
fmt.Println(message1)
```

Penerimaan channel bersifat blocking. Artinya statement `var message1 = <-messages` hingga setelahnya tidak akan dieksekusi sebelum ada data yang dikirim lewat channel.

Ketiga goroutine tersebut datanya akan diterima secara berurutan oleh `message1`, `message2`, `message3`; untuk kemudian ditampilkan.

```
[novalagung:belajar-golang $ go run bab30.go
hello wick
hello bourne
hello hunt
[novalagung:belajar-golang $ go run bab30.go
hello wick
hello hunt
hello bourne
[novalagung:belajar-golang $ go run bab30.go
hello hunt
hello bourne
hello wick
novalagung:belajar-golang $ ]]
```

Dari screenshot output di atas bisa dilihat bahwa text yang dikembalikan oleh `sayHelloTo` tidak selalu berurutan, meskipun penerimaan datanya adalah berurutan. Hal ini dikarenakan, pengiriman data adalah dari 3 goroutine yang berbeda, yang kita tidak tau mana yang prosesnya selesai lebih dulu. Goroutine yang dieksekusi lebih awal belum tentu selesai lebih awal, yang jelas proses yang selesai lebih awal datanya akan diterima lebih awal.

Karena pengiriman dan penerimaan data lewat channel bersifat **blocking**, tidak perlu memanfaatkan sifat blocking dari fungsi `fmt.Scanln()` untuk mengantisipasi goroutine utama `main` selesai sebelum 3 goroutine di atas selesai.

## A.30.2. Channel Sebagai Tipe Data Parameter

Variabel channel bisa di-passing ke fungsi lain sebagai parameter. Caranya dengan menambahkan keyword `chan` ketika deklarasinya.

Siapkan fungsi `printMessage` dengan parameter adalah channel. Lalu ambil data yang dikirimkan lewat channel tersebut untuk ditampilkan.

```
func printMessage(what chan string) {
    fmt.Println(<-what)
}
```

Setelah itu ubah implementasi di fungsi `main`.

```
func main() {
    runtime.GOMAXPROCS(2)

    var messages = make(chan string)

    for _, each := range []string{"wick", "hunt", "bourne"} {
        go func(who string) {
            var data = fmt.Sprintf("hello %s", who)
            messages <- data
        }(each)
    }

    for i := 0; i < 3; i++ {
        printMessage(messages)
    }
}
```

Parameter `what` fungsi `printMessage` bertipe channel `string`, bisa dilihat dari kode `chan string` pada cara deklarasinya. Operasi serah-terima data akan bisa dilakukan pada variabel tersebut, dan akan berdampak juga pada variabel `messages` di fungsi `main`.

Passing data bertipe channel lewat parameter secara implisit adalah **pass by reference**, yang di-passing adalah pointer-nya. Output program di atas adalah sama dengan program sebelumnya.

```
[novalagung:belajar-golang $ go run bab30.go
hello hunt
hello bourne
hello wick
[novalagung:belajar-golang $ go run bab30.go
hello wick
hello bourne
hello hunt
novalagung:belajar-golang $ ]
```

Berikut merupakan penjelasan tambahan kode di atas.

### A.30.3. Iterasi Data Array Langsung Pada Saat Inisialisasi

Data array yang baru di-inisialisasi bisa langsung di-iterasi, caranya mudah dengan menuliskannya langsung setelah keyword `range`.

```
for _, each := range []string{"wick", "hunt", "bourne"} {
    // ...
}
```

## A.30.4. Eksekusi Goroutine Pada IIFE

Eksekusi goroutine tidak harus pada fungsi atau closure yang sudah terdefinisi. Sebuah IIFE juga bisa dijalankan sebagai goroutine baru. Caranya dengan langsung menambahkan keyword `go` pada waktu deklarasi-eksekusi IIFE-nya.

```
var messages = make(chan string)

go func(who string) {
    var data = fmt.Sprintf("hello %s", who)
    messages <- data
}("wick")

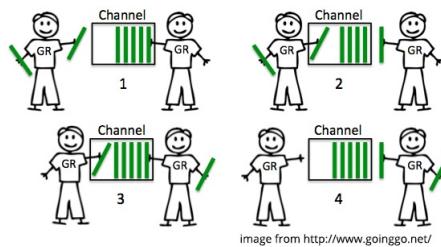
var message = <-messages
fmt.Println(message)
```

## A.31. Buffered Channel

Channel secara default adalah **un-buffered**, tidak di-buffer di memori. Ketika ada goroutine yang mengirimkan data lewat channel, harus ada goroutine lain yang bertugas menerima data dari channel yang sama, dengan proses serah-terima yang bersifat blocking. Maksudnya, baris kode di bagian pengiriman dan penerimaan data, tidak akan akan diproses sebelum proses serah-terimanya selesai.

Buffered channel sedikit berbeda. Pada channel jenis ini, ditentukan jumlah buffer-nya. Angka tersebut menjadi penentu jumlah data yang dikirimkan bersamaan. Selama jumlah data yang dikirim tidak melebihi jumlah buffer, maka pengiriman akan berjalan **asynchronous** (tidak blocking).

Ketika jumlah data yang dikirim sudah melewati batas buffer, maka pengiriman data hanya bisa dilakukan ketika salah satu data sudah diambil dari channel, sehingga ada slot channel yang kosong. Dengan proses penerimaannya sendiri bersifat blocking.



### A.31.1. Penerapan Buffered Channel

Penerapan buffered channel pada dasarnya mirip seperti channel biasa. Perbedannya pada channel jenis ini perlu disiapkan jumlah buffer-nya.

Berikut adalah contoh penerapan buffered channel. Program dibawah ini merupakan pembuktian bahwa pengiriman data lewat buffered channel adalah asynchronous selama jumlah data yang sedang di-buffer oleh channel tidak melebihi kapasitas buffernya.

```
package main

import "fmt"
import "runtime"

func main() {
    runtime.GOMAXPROCS(2)

    messages := make(chan int, 2)

    go func() {
        for {
            i := <-messages
            fmt.Println("receive data", i)
        }
    }()

    for i := 0; i < 5; i++ {
        fmt.Println("send data", i)
        messages <- i
    }
}
```

Pada kode di atas, parameter kedua fungsi `make` adalah representasi jumlah buffer. Perlu diperhatikan bahwa nilai buffered channel dimulai dari `0`. Ketika nilainya adalah **2**, brarti jumlah buffer maksimal ada **3** (0, 1, dan 2).

Pada contoh di atas, terdapat juga sebuah goroutine yang berisikan proses penerimaan data dari channel message, yang selanjutnya akan ditampilkan.

Setelah goroutine untuk penerimaan data dieksekusi, data dikirimkan lewat perulangan `for`. Sejumlah 5 data akan dikirim lewat channel `message` secara sekuensial.

```
[novalagung:belajar-golang $ go run bab31.go
send data 0
send data 1
send data 2
send data 3
receive data 0
receive data 1
receive data 2
send data 4
receive data 3
receive data 4
[novalagung:belajar-golang $ go run bab31.go
send data 0
send data 1
send data 2
send data 3
receive data 0
receive data 1
receive data 2
send data 4
novalagung:belajar-golang $ ]
```

Bisa dilihat hasilnya pada output di atas. Pengiriman data ke-4, diikuti dengan penerimaan data, dan kedua proses tersebut berjalan secara blocking.

Pengiriman data ke 0, 1, 2 dan 3 akan berjalan secara asynchronous, hal ini karena channel ditentukan nilai buffer-nya sebanyak 3 (ingat, dimulai dari 0). Pengiriman selanjutnya (ke-4 dan ke-5) hanya akan terjadi jika ada salah satu data dari 4 data yang sebelumnya telah dikirimkan, sudah diterima (dengan serah terima data yang bersifat blocking). Setelahnya, sesudah slot channel ada yang kosong, serah-terima akan kembali asynchronous.

## A.32. Channel - Select

Adanya channel memang sangat membantu pengontrolan goroutine, jumlah goroutine yang banyak bukan lagi masalah.

Fungsi utama channel bukan untuk mengontrol goroutine, melainkan untuk sharing data antar goroutine.

Namun channel memang bisa digunakan untuk mengontrol goroutine.

Ada kalanya dimana kita butuh tak hanya satu channel saja untuk handle komunikasi data pada goroutine yang jumlahnya juga banyak, dibutuhkan beberapa atau mungkin banyak channel.

Disinilah kegunaan dari `select`. Select memudahkan pengontrolan komunikasi data lewat channel. Cara penggunaannya sama seperti seleksi kondisi `switch`.

### A.32.1. Penerapan Keyword `select`

Program pencarian rata-rata dan nilai tertinggi berikut merupakan contoh sederhana penerapan select dalam channel. Akan ada 2 buah goroutine yang masing-masing di-handle oleh sebuah channel. Setiap kali goroutine selesai dieksekusi, akan dikirimkan datanya ke channel yang bersangkutan. Lalu dengan menggunakan select, akan diatur penerimaan datanya.

Pertama, kita siapkan terlebih dahulu 2 fungsi yang akan dieksekusi sebagai goroutine baru. Fungsi pertama digunakan untuk mencari rata-rata, dan fungsi kedua untuk penentuan nilai tertinggi dari sebuah slice.

```
package main

import "fmt"
import "runtime"

func getAverage(numbers []int, ch chan float64) {
    var sum = 0
    for _, e := range numbers {
        sum += e
    }
    ch <- float64(sum) / float64(len(numbers))
}

func getMax(numbers []int, ch chan int) {
    var max = numbers[0]
    for _, e := range numbers {
        if max < e {
            max = e
        }
    }
    ch <- max
}
```

Kedua fungsi di atas nantinya dijalankan sebagai goroutine baru. Di akhir masing-masing fungsi, dikirimkan data hasil komputasi ke channel yang sudah ditentukan (`ch1` menampung data rata-rata, `ch2` untuk data nilai tertinggi).

Buat implementasinya pada fungsi `main`.

```
func main() {
    runtime.GOMAXPROCS(2)

    var numbers = []int{3, 4, 3, 5, 6, 3, 2, 2, 6, 3, 4, 6, 3}
    fmt.Println("numbers :", numbers)
```

```

var ch1 = make(chan float64)
go getAverage(numbers, ch1)

var ch2 = make(chan int)
go getMax(numbers, ch2)

for i := 0; i < 2; i++ {
    select {
    case avg := <-ch1:
        fmt.Printf("Avg \t: %.2f \n", avg)
    case max := <-ch2:
        fmt.Printf("Max \t: %d \n", max)
    }
}
}

```

Pada kode di atas, transaksi pengiriman data pada channel `ch1` dan `ch2` dikontrol menggunakan `select`.

Terdapat 2 buah `case` kondisi penerimaan data dari kedua channel tersebut.

- Kondisi `case avg := <-ch1` akan terpenuhi ketika ada penerimaan data dari channel `ch1`, yang kemudian akan ditampung oleh variabel `avg`.
- Kondisi `case max := <-ch2` akan terpenuhi ketika ada penerimaan data dari channel `ch2`, yang kemudian akan ditampung oleh variabel `max`.

Karena ada 2 buah channel, maka perlu disiapkan perulangan 2 kali sebelum penggunaan keyword `select`.

```

[novalagung:belajar-golang $ go run bab32.go
numbers : [3 4 3 5 6 3 2 2 6 3 4 6 3]
Avg      : 3.85
Max      : 6
[novalagung:belajar-golang $ go run bab32.go
numbers : [3 4 3 5 6 3 2 2 6 3 4 6 3]
Max      : 6
Avg      : 3.85
novalagung:belajar-golang $ ]

```

Cukup mudah bukan?

## A.33. Channel - Range dan Close

Penerimaan data lewat channel yang jumlah goroutine-nya banyak bisa lebih mudah dengan memanfaatkan keyword `for - range`.

`for - range` jika diterapkan pada channel berfungsi untuk handle penerimaan data. Setiap kali ada pengiriman data ke channel, perulangan akan berjalan sekali, setelah 1 perulangan selesai akan menunggu lagi penerimaan data selanjutnya. Perulangan hanya akan berhenti jika channel di `close` atau di-non-aktifkan. Fungsi `close` digunakan untuk me-non-aktifkan channel.

Channel yang sudah di-close tidak bisa digunakan lagi untuk menerima maupun mengirim data, itulah kenapa perulangan pada `for - range` juga ikut berhenti.

### A.33.1. Penerapan `for - range - close` Pada Channel

Berikut adalah contoh program yang menggunakan `for - range` untuk penerimaan data dari channel.

Pertama siapkan fungsi `sendMessage()` bertugas untuk mengirim data. Didalam fungsi ini dijalankan perulangan sebanyak 20 kali, di tiap perulangannya data dikirim lewat channel. Setelah semua data terkirim, channel di-close.

```
func sendMessage(ch chan<- string) {
    for i := 0; i < 20; i++ {
        ch <- fmt.Sprintf("data %d", i)
    }
    close(ch)
}
```

Siapkan juga fungsi `printMessage()` untuk handle penerimaan data. Didalamnya, channel akan di-looping menggunakan `for - range`, yang kemudian ditampilkan data-nya.

```
func printMessage(ch <-chan string) {
    for message := range ch {
        fmt.Println(message)
    }
}
```

Buat channel baru dalam `main`, jalankan `sendMessage()` sebagai goroutine. Jalankan juga `printMessage()`. Dengan ini 20 data dikirimkan lewat goroutine baru, dan nantinya diterima di goroutine utama.

```
func main() {
    runtime.GOMAXPROCS(2)

    var messages = make(chan string)
    go sendMessage(messages)
    printMessage(messages)
}
```

Setelah 20 data sukses dikirim dan diterima, channel `ch` di-non-aktifkan (`close(ch)`). Membuat perulangan data channel dalam `printMessage()` juga akan berhenti.

```
[novalagung:belajar-golang $ go run bab33.go
data 0
data 1
data 2
data 3
data 4
data 5
data 6
data 7
data 8
data 9
data 10
data 11
data 12
data 13
data 14
data 15
data 16
data 17
data 18
data 19
novalagung:belajar-golang $ ]
```

Berikut adalah penjelasan tambahan mengenai channel.

### A.33.2. Channel Direction

Ada yang unik dengan fitur parameter channel yang disediakan Golang. Level akses channel bisa ditentukan, apakah hanya sebagai penerima, pengirim, atau penerima sekaligus pengirim. Konsep ini disebut dengan **channel direction**.

Cara pemberian level akses adalah dengan menambahkan tanda `<-` sebelum atau setelah keyword `chan`. Untuk lebih jelasnya bisa dilihat di list berikut.

Sintaks	Penjelasan
<code>ch chan string</code>	Parameter <code>ch</code> bisa digunakan untuk <b>mengirim</b> dan <b>menerima</b> data
<code>ch chan&lt;- string</code>	Parameter <code>ch</code> hanya bisa digunakan untuk <b>mengirim</b> data
<code>ch &lt;-chan string</code>	Parameter <code>ch</code> hanya bisa digunakan untuk <b>menerima</b> data

Pada kode di atas bisa dilihat bahwa secara default channel akan memiliki kemampuan untuk mengirim dan menerima data. Untuk mengubah channel tersebut agar hanya bisa mengirim atau menerima saja, dengan memanfaatkan simbol `<-`.

Sebagai contoh fungsi `sendMessage(ch chan<- string)` yang parameter `ch` dideklarasikan dengan level akses untuk pengiriman data saja. Channel tersebut hanya bisa digunakan untuk mengirim, contohnya: `ch <- fmt.Sprintf("data %d", i)`.

Dan sebaliknya pada fungsi `printMessage(ch <-chan string)`, channel `ch` hanya bisa digunakan untuk menerima data saja.

## A.34. Channel - Timeout

Teknik timeout digunakan untuk mengontrol penerimaan data dari channel berdasarkan waktu diterimanya, dengan durasi timeout bisa kita tentukan sendiri.

Ketika tidak ada aktivitas penerimaan data dalam durasi yang sudah ditentukan, callback akan dijalankan.

### A.34.1. Penerapan Channel Timeout

Berikut adalah program sederhana tentang pengaplikasian timeout pada channel. Sebuah goroutine baru dijalankan dengan tugas mengirimkan data setiap interval tertentu, dengan durasi interval-nya adalah acak/random.

```
package main

import "fmt"
import "math/rand"
import "runtime"
import "time"

func sendData(ch chan<- int) {
    for i := 0; true; i++ {
        ch <- i
        time.Sleep(time.Duration(rand.Int()%10+1) * time.Second)
    }
}
```

Selanjutnya, disiapkan perulangan tanpa henti, yang di tiap perulangannya ada seleksi kondisi channel menggunakan `select`.

```
func retreiveData(ch <-chan int) {
loop:
for {
    select {
    case data := <-ch:
        fmt.Print(`receive data ``, data, `", "\n")
    case <-time.After(time.Second * 5):
        fmt.Println("timeout. no activities under 5 seconds")
        break loop
    }
}
}
```

Ada 2 blok kondisi pada `select` tersebut.

- `case data := <-messages:`, akan terpenuhi ketika ada serah terima data pada channel `messages`
- `case <-time.After(time.Second * 5):`, akan terpenuhi ketika tidak ada aktivitas penerimaan data dari channel dalam durasi 5 detik. Blok inilah yang kita sebut sebagai callback.

Terakhir, kedua fungsi tersebut dipanggil di `main`.

```
func main() {
    rand.Seed(time.Now().Unix())
    runtime.GOMAXPROCS(2)

    var messages = make(chan int)

    go sendData(messages)
```

```
    retreiveData(messages)
}
```

Muncul output setiap kali ada penerimaan data dengan delay waktu acak. Ketika tidak ada aktifitas penerimaan dari channel dalam durasi 5 detik, perulangan pengecekan channel diberhentikan.

```
[novalagung:belajar-golang $ go run bab34.go
receive data "0"
receive data "1"
receive data "2"
receive data "3"
timeout. no activities under 5 seconds
[novalagung:belajar-golang $ go run bab34.go
receive data "0"
timeout. no activities under 5 seconds
[novalagung:belajar-golang $ go run bab34.go
receive data "0"
timeout. no activities under 5 seconds
[novalagung:belajar-golang $ go run bab34.go
receive data "0"
receive data "1"
timeout. no activities under 5 seconds
novalagung:belajar-golang $ ]
```

## A.35. Defer & Exit

**Defer** digunakan untuk mengakhirkan eksekusi sebuah statement. Sedangkan **Exit** digunakan untuk menghentikan program (ingat, menghentikan program, tidak seperti `return` yang menghentikan blok kode).

### A.35.1. Penerapan keyword defer

Seperti yang sudah dijelaskan secara singkat di atas, bahwa defer digunakan untuk mengakhirkan eksekusi baris kode. Ketika eksekusi blok sudah selesai, statement yang di defer baru akan dijalankan.

Defer bisa ditempatkan di mana saja, awal maupun akhir blok.

```
package main

import "fmt"

func main() {
    defer fmt.Println("halo")
    fmt.Println("selamat datang")
}
```

Output:

```
[novalagung:belajar-golang $ go run bab35.go
selamat datang
halo
novalagung:belajar-golang $ ]
```

Keyword `defer` digunakan untuk mengakhirkan statement. Pada kode di atas, `fmt.Println("halo")` di-defer, hasilnya string `"halo"` akan muncul setelah `"selamat datang"`.

Statement yang di-defer akan tetap muncul meskipun blok kode diberhentikan ditengah jalan menggunakan `return`. Contohnya seperti pada kode berikut.

```
func main() {
    orderSomeFood("pizza")
    orderSomeFood("burger")
}

func orderSomeFood(menu string) {
    defer fmt.Println("Terimakasih, silakan tunggu")
    if menu == "pizza" {
        fmt.Print("Pilihan tepat!", " ")
        fmt.Print("Pizza ditempat kami paling enak!", "\n")
        return
    }
    fmt.Println("Pesanan anda:", menu)
}
```

Output:

```
[novalagung:chapter-35 $ go run 2-defer-return.go
Pilihan tepat! Pizza ditempat kami paling enak!
Terimakasih, silakan tunggu
Pesanan anda: burger
Terimakasih, silakan tunggu
novalagung:chapter-35 $ ]
```

Info tambahan, ketika ada banyak statement yang di-defer, maka kesemuanya akan dieksekusi di akhir secara berurutan.

## A.35.2. Penerapan Fungsi `os.Exit()`

Exit digunakan untuk menghentikan program secara paksa pada saat itu juga. Semua statement setelah exit tidak akan di eksekusi, termasuk juga defer.

Fungsi `os.Exit()` berada dalam package `os`. Fungsi ini memiliki sebuah parameter bertipe numerik yang wajib diisi. Angka yang dimasukkan akan muncul sebagai **exit status** ketika program berhenti.

```
package main

import "fmt"
import "os"

func main() {
    defer fmt.Println("halo")
    os.Exit(1)
    fmt.Println("selamat datang")
}
```

Meskipun `defer fmt.Println("halo")` ditempatkan sebelum `os.Exit()`, statement tersebut tidak akan dieksekusi, karena di-tengah fungsi program dihentikan secara paksa.

```
[novalagung:belajar-golang $ go run bab35.go
exit status 1
novalagung:belajar-golang $ ]
```

## A.36. Error, Panic, dan Recover

Error merupakan topik yang penting dalam pemrograman go. Di bagian ini kita akan belajar mengenai pemanfaatan error dan cara membuat custom error sendiri.

Kita juga akan belajar tentang penggunaan `panic` untuk memunculkan panic error, dan `recover` untuk mengatasinya.

### A.36.1. Pemanfaatan Error

`error` adalah sebuah tipe. Error memiliki memiliki 1 buah property berupa method `Error()`, method ini mengembalikan detail pesan error. Error termasuk tipe yang isinya bisa kosong atau `nil`.

Di go, banyak sekali fungsi yang mengembalikan nilai balik lebih dari satu. Biasanya, salah satu kembalian adalah bertipe `error`. Contohnya seperti pada fungsi `strconv.Atoi()`.

`strconv.Atoi()` berguna untuk mengkonversi data string menjadi numerik. Fungsi ini mengembalikan 2 nilai balik. Nilai balik pertama adalah hasil konversi, dan nilai balik kedua adalah `error`.

Ketika konversi berjalan mulus, nilai balik kedua akan bernilai `nil`. Sedangkan ketika konversi gagal, penyebabnya bisa langsung diketahui dari nilai balik kedua.

Dibawah ini merupakan contoh program sederhana untuk deteksi inputan dari user, apakah numerik atau bukan. Dari sini kita akan belajar mengenai pemanfaatan error.

```
package main

import (
    "fmt"
    "strconv"
)

func main() {
    var input string
    fmt.Println("Type some number: ")
    fmt.Scanln(&input)

    var number int
    var err error
    number, err = strconv.Atoi(input)

    if err == nil {
        fmt.Println(number, "is number")
    } else {
        fmt.Println(input, "is not number")
        fmt.Println(err.Error())
    }
}
```

Ketika program dijalankan, muncul tulisan `"Type some number: "`, ketik sebuah angka lalu enter.

`fmt.Scanln(&input)` bertugas mengambil inputan yang diketik user sebelum dia menekan enter, lalu menyimpannya sebagai string ke variabel `input`.

Selanjutnya variabel tersebut dikonversi ke tipe numerik menggunakan `strconv.Atoi()`. Fungsi tersebut mengembalikan 2 data, ditampung oleh `number` dan `err`.

Data pertama (`number`) berisi hasil konversi. Dan data kedua `err`, berisi informasi errornya (jika memang terjadi error ketika proses konversi).

Setelah itu dilakukan pengecekan, ketika tidak ada error, `number` ditampilkan. Dan jika ada error, `input` ditampilkan beserta pesan errornya.

Pesan error bisa didapat dari method `Error()` milik tipe `error`.

```
[novalagung:belajar-golang $ go run bab36.go
Type some number: 24
24 is number
[novalagung:belajar-golang $ go run bab36.go
Type some number: 2a
2a is not number
strconv.ParseInt: parsing "2a": invalid syntax
novalagung:belajar-golang $ ]
```

## A.36.2. Membuat Custom Error

Selain memanfaatkan error hasil kembalian fungsi, kita juga bisa membuat error sendiri dengan menggunakan fungsi `errors.New` (untuk menggunakannya harus import package `errors` terlebih dahulu).

Pada contoh berikut ditunjukkan bagaimana cara membuat custom error. Pertama siapkan fungsi dengan nama `validate()`, yang nantinya digunakan untuk pengecekan input, apakah inputan kosong atau tidak. Ketika kosong, maka error baru akan dibuat.

```
package main

import (
    "errors"
    "fmt"
    "strings"
)

func validate(input string) (bool, error) {
    if strings.TrimSpace(input) == "" {
        return false, errors.New("cannot be empty")
    }
    return true, nil
}
```

Selanjutnya di fungsi main, buat proses sederhana untuk capture inputan user. Manfaatkan fungsi `validate()` untuk mengecek inputannya.

```
func main() {
    var name string
    fmt.Print("Type your name: ")
    fmt.Scanln(&name)

    if valid, err := validate(name); valid {
        fmt.Println("halo", name)
    } else {
        fmt.Println(err.Error())
    }
}
```

Fungsi `validate()` mengembalikan 2 data. Data pertama adalah nilai `bool` yang menandakan inputan apakah valid atau tidak. Data ke-2 adalah pesan error-nya (jika inputan tidak valid).

Fungsi `strings.TrimSpace()` digunakan untuk menghilangkan karakter spasi sebelum dan sesudah string. Ini dibutuhkan karena user bisa saja menginputkan spasi lalu enter.

Ketika inputan tidak valid, maka error baru dibuat dengan memanfaatkan fungsi `errors.New()`.

```
[novalagung:belajar-golang $ go run bab36.go
Type your name: wick
halo wick
[novalagung:belajar-golang $ go run bab36.go
Type your name:
cannot be empty
novalagung:belajar-golang $ ]
```

### A.36.3. Penggunaan panic

Panic digunakan untuk menampilkan *trace* error sekaligus menghentikan flow goroutine (ingat, `main` juga merupakan goroutine). Setelah ada panic, proses selanjutnya tidak di-eksekusi (kecuali proses yang di-defer, akan tetap dijalankan tepat sebelum panic muncul).

Panic memnuculkan pesan di console, sama seperti `fmt.Println()` hanya saja informasi yang ditampilkan sangat mendetail dan heboh.

Pada program yang telah kita buat tadi, ubah `fmt.Println()` yang berada di dalam blok kondisi `else` pada fungsi `main` menjadi `panic()`, lalu tambahkan `fmt.Println()` setelahnya.

```
func main() {
    var name string
    fmt.Print("Type your name: ")
    fmt.Scanln(&name)

    if valid, err := validate(name); valid {
        fmt.Println("halo", name)
    } else {
        fmt.Println(err.Error())
        fmt.Println("end")
    }
}
```

Coba jalankan program, lalu langsung tekan enter, error panic akan muncul, dan baris kode setelahnya tidak dijalankan.

```
[novalagung:belajar-golang $ go run bab36.go
Type your first name: wick
halo wick
[novalagung:belajar-golang $ go run bab36.go
Type your first name:
panic: cannot be empty

goroutine 1 [running]:
main.main()
    /Users/novalagung/Documents/go/src/belajar-golang/bab36.go:26 +0x37c
exit status 2
novalagung:belajar-golang $ ]
```

### A.36.4. Penggunaan recover

Recover berguna untuk meng-handle panic error. Pada saat panic muncul, recover men-take-over goroutine yang sedang panic (pesan panic tidak akan muncul).

Kita modif sedikit fungsi di-atas untuk mempraktekkan bagaimana cara penggunaan recover.

Tambahkan fungsi `catch()`, dalam fungsi ini terdapat statement `recover()` yang dia akan mengembalikan pesan error panic yang seharusnya muncul. Fungsi yang berisikan `recover()` harus di-defer, karena meskipun muncul panic, defer tetap di-eksekusi.

```
func catch() {
    if r := recover(); r != nil {
```

```

        fmt.Println("Error occured", r)
    } else {
        fmt.Println("Application running perfectly")
    }
}

func main() {
    defer catch()

    var name string
    fmt.Print("Type your name: ")
    fmt.Scanln(&name)

    if valid, err := validate(name); valid {
        fmt.Println("halo", name)
    } else {
        panic(err.Error())
        fmt.Println("end")
    }
}

```

Output:

```

[novalagung:chapter-36 $ go run 4-recover.go
Type your name: john
halo john
Application running perfectly
[novalagung:chapter-36 $
[novalagung:chapter-36 $ go run 4-recover.go
Type your name:
Error occured cannot be empty
[novalagung:chapter-36 $ ]

```

## A.36.5. Pemanfaatan recover pada IIFE

Recover akan dieksekusi (pada waktunya) di blok kode dimana blok fungsi untuk recover ditempatkan.

```

func main() {

    // recover untuk panic dalam fungsi main
    defer func() {
        if r := recover(); r != nil {
            fmt.Println("Panic occured", r)
        } else {
            fmt.Println("Application running perfectly")
        }
    }()

    panic("some error happen")
}

```

Di real-world development, ada kalanya recover dibutuhkan tidak pada fungsi terluar, tetapi dalam blok kode yg lebih spesifik. Sebagai contoh: recover panic dalam perulangan tanpa memberhentikan perulangan itu sendiri.

```

func main() {
    data := []string{"superman", "aquaman", "wonder woman"}

    for _, each := range data {

        func() {

            // recover untuk iife dalam perulangan
            defer func() {

```

```
        if r := recover(); r != nil {
            fmt.Println("Panic occurred on looping", each, "| message:", r)
        } else {
            fmt.Println("Application running perfectly")
        }
    }()
}
panic("some error happen")
}()
```

Pada kode di atas, di dalam perulangan terdapat sebuah IIFE. Berisi operasi sesuai kebutuhan dan sebuah fungsi yang di-defer untuk recover panic. Ketika terjadi panic, maka perulangan tersebut tidak akan rusak, tetap lanjut.

## A.37. Layout Format String

Di bab-bab sebelumnya kita telah banyak menggunakan layout format string seperti `%s`, `%d`, `%.2f`, dan lainnya; untuk keperluan menampilkan output ke layar ataupun untuk memformat string.

Layout format string digunakan dalam konversi data ke bentuk string. Contohnya seperti `%.3f` yang untuk konversi nilai `double` ke `string` dengan 3 digit desimal.

Pada bab ini kita akan mempelajari satu per satu layout format string yang tersedia di Golang. Kode berikut adalah sampel data yang akan kita gunakan sebagai contoh.

```
type student struct {
    name      string
    height    float64
    age       int32
    isGraduated bool
    hobbies   []string
}

var data = student{
    name:      "wick",
    height:    182.5,
    age:       26,
    isGraduated: false,
    hobbies:   []string{"eating", "sleeping"},
}
```

### A.37.1. Layout Format `%b`

Digunakan untuk memformat data numerik, menjadi bentuk string numerik berbasis 2 (biner).

```
fmt.Printf("%b\n", data.age)
// 11010
```

### A.37.2. Layout Format `%c`

Digunakan untuk memformat data numerik yang merupakan kode unicode, menjadi bentuk string karakter unicode-nya.

```
fmt.Printf("%c\n", 1400)
// n

fmt.Printf("%c\n", 1235)
// ä
```

### A.37.3. Layout Format `%d`

Digunakan untuk memformat data numerik, menjadi bentuk string numerik berbasis 10 (basis bilangan yang kita gunakan).

```
fmt.Printf("%d\n", data.age)
// 26
```

## A.37.4. Layout Format `%e` atau `%E`

Digunakan untuk memformat data numerik desimal ke dalam bentuk notasi numerik standar [Scientific notation](#).

```
fmt.Printf("%e\n", data.height)
// 1.825000e+02

fmt.Printf("%E\n", data.height)
// 1.825000E+02
```

**1.825000E+02** maksudnya adalah  $1.825 \times 10^2$ , dan hasil operasi tersebut adalah sesuai dengan data asli = **182.5**.

Perbedaan antara `%e` dan `%E` hanya pada bagian huruf besar kecil karakter `e` pada hasil.

## A.37.5. Layout Format `%f` atau `%F`

`%F` adalah alias dari `%f`. Keduanya memiliki fungsi yang sama.

Berfungsi untuk memformat data numerik desimal, dengan lebar desimal bisa ditentukan. Secara default lebar digit desimal adalah 6 digit.

```
fmt.Printf("%f\n", data.height)
// 182.500000

fmt.Printf("%.9f\n", data.height)
// 182.500000000

fmt.Printf("%.2f\n", data.height)
// 182.50

fmt.Printf("%.f\n", data.height)
// 182
```

## A.37.6. Layout Format `%g` atau `%G`

`%G` adalah alias dari `%g`. Keduanya memiliki fungsi yang sama.

Berfungsi untuk memformat data numerik desimal, dengan lebar desimal bisa ditentukan. Lebar kapasitasnya sangat besar, pas digunakan untuk data yang jumlah digit desimalnya cukup banyak.

Bisa dilihat pada kode berikut perbandingan antara `%e`, `%f`, dan `%g`.

```
fmt.Printf("%e\n", 0.123123123123)
// 1.231231e-01

fmt.Printf("%f\n", 0.123123123123)
// 0.123123

fmt.Printf("%g\n", 0.123123123123)
// 0.123123123123
```

Perbedaan lainnya adalah pada `%g`, lebar digit desimal adalah sesuai dengan datanya, tidak bisa dicustom seperti pada `%f`.

```
fmt.Printf("%g\n", 0.12)
// 0.12
```

```
fmt.Printf("%g\n", 0.12)
// 0.12
```

## A.37.7. Layout Format %o

Digunakan untuk memformat data numerik, menjadi bentuk string numerik berbasis 8 (oktal).

```
fmt.Printf("%o\n", data.age)
// 32
```

## A.37.8. Layout Format %p

Digunakan untuk memformat data pointer, mengembalikan alamat pointer referensi variabelnya.

Alamat pointer dituliskan dalam bentuk numerik berbasis 16 dengan prefix `0x`.

```
fmt.Printf("%p\n", &data.name)
// 0x2081be0c0
```

## A.37.9. Layout Format %q

Digunakan untuk **escape** string. Meskipun string yang dipakai menggunakan literal `\` akan tetap di-escape.

```
fmt.Printf("%q\n", ` name \ height `)
// " name \\ height "
```

## A.37.10. Layout Format %s

Digunakan untuk memformat data string.

```
fmt.Printf("%s\n", data.name)
// wick
```

## A.37.11. Layout Format %t

Digunakan untuk memformat data boolean, menampilkan nilai `bool`-nya.

```
fmt.Printf("%t\n", data.isGraduated)
// false
```

## A.37.12. Layout Format %T

Berfungsi untuk mengambil tipe variabel yang akan diformat.

```
fmt.Printf("%T\n", data.name)
// string

fmt.Printf("%T\n", data.height)
```

```
// float64
fmt.Printf("%T\n", data.age)
// int32

fmt.Printf("%T\n", data.isGraduated)
// bool

fmt.Printf("%T\n", data.hobbies)
// []string
```

## A.37.13. Layout Format %v

Digunakan untuk memformat data apa saja (termasuk data bertipe `interface{}`). Hasil kembalinya adalah string nilai data aslinya.

Jika data adalah objek cetakan `struct`, maka akan ditampilkan semua secara property berurutan.

```
fmt.Printf("%v\n", data)
// {wick 182.5 26 false [eating sleeping]}
```

## A.37.14. Layout Format %+v

Digunakan untuk memformat struct, mengembalikan nama tiap property dan nilainya berurutan sesuai dengan struktur struct.

```
fmt.Printf("%+v\n", data)
// {name:wick height:182.5 age:26 isGraduated:false hobbies:[eating sleeping]}
```

## A.37.15. Layout Format %#v

Digunakan untuk memformat struct, mengembalikan nama dan nilai tiap property sesuai dengan struktur struct dan juga bagaimana objek tersebut dideklarasikan.

```
fmt.Printf("%#v\n", data)
// main.student{name:"wick", height:182.5, age:26, isGraduated:false, hobbies:[]string{"eating", "sleeping"}}
```

Ketika menampilkan objek yang deklarasinya adalah menggunakan teknik *anonymous struct*, maka akan muncul juga struktur anonymous struct nya.

```
var data = struct {
    name   string
    height float64
} {
    name:   "wick",
    height: 182.5,
}

fmt.Printf("%#v\n", data)
// struct { name string; height float64 }{name:"wick", height:182.5}
```

Format ini juga bisa digunakan untuk menampilkan tipe data lain, dan akan dimunculkan strukturnya juga.

## A.37.16. Layout Format `%x` atau `%X`

Digunakan untuk memformat data numerik, menjadi bentuk string numerik berbasis 16 (heksadesimal).

```
fmt.Printf("%x\n", data.age)
// 1a
```

Jika digunakan pada tipe data string, maka akan mengembalikan kode heksadesimal tiap karakter.

```
var d = data.name

fmt.Printf("%x%x%x%x\n", d[0], d[1], d[2], d[3])
// 7769636b

fmt.Printf("%X\n", d)
// 7769636b
```

`%x` dan `%X` memiliki fungsi yang sama. Perbedaannya adalah `%x` akan mengembalikan string dalam bentuk uppercase atau huruf kapital.

## A.37.17. Layout Format `%%`

Cara untuk menulis karakter `%` pada string format.

```
fmt.Printf("%%\n")
// %
```

## A.8. Time, Parsing Time, & Format Time

Pada bab ini kita akan belajar tentang pemanfaatan data bertipe waktu, method-method yang disediakan, dan juga **format & parsing** data `string` ke tipe `time.Time` dan sebaliknya.

Golang menyediakan package `time` yang berisikan banyak sekali komponen yang bisa digunakan untuk keperluan pemanfaatan waktu. Salah satunya adalah `time.Time`, yang merupakan tipe untuk data tanggal dan waktu di Golang.

Time disini maksudnya adalah gabungan `date` dan `time`, bukan hanya waktu saja.

### A.8.1. Penggunaan `time.Time`

`time.Time` adalah tipe data untuk objek waktu. Ada 2 cara yang bisa digunakan untuk membuat data bertipe ini.

- Menjadikan informasi waktu sekarang sebagai objek `time.Time`.
- Membuat objek baru bertipe `time.Time` dengan informasi ditentukan sendiri.

Berikut merupakan contoh penggunannya.

```
package main

import "fmt"
import "time"

func main() {
    var time1 = time.Now()
    fmt.Printf("time1 %v\n", time1)
    // time1 2015-09-01 17:59:31.73600891 +0700 WIB

    var time2 = time.Date(2011, 12, 24, 10, 20, 0, 0, time.UTC)
    fmt.Printf("time2 %v\n", time2)
    // time2 2011-12-24 10:20:00 +0000 UTC
}
```

Fungsi `time.Now()` mengembalikan objek `time.Time` dengan informasi adalah waktu sekarang, waktu tepat ketika statement tersebut dijalankan. Bisa dilihat ketika di tampilkan, informasi yang muncul adalah sesuai dengan tanggal program tersebut dieksekusi.

```
[novalagung:belajar-golang $ go run bab38.go
time1 2015-10-08 10:02:43.584690264 +0700 WIB
time2 2011-12-24 10:20:00 +0000 UTC
novalagung:belajar-golang $ ]
```

Fungsi `time.Date()` digunakan untuk membuat objek `time.Time` baru yang informasi waktunya kita tentukan sendiri. Fungsi ini memiliki 8 buah parameter **mandatory** dengan skema bisa dilihat di kode berikut:

```
time.Date(tahun, bulan, tanggal, jam, menit, detik, nanodetik, timezone)
```

Objek cetakan fungsi `time.Now()`, informasi timezone-nya adalah relatif terhadap lokasi kita. Karena kebetulan penulis berlokasi di Jawa Timur, maka akan terdeteksi masuk dalam **GMT+7** atau **WIB**. Berbeda dengan variabel `time2` yang lokasinya sudah kita tentukan secara eksplisit yaitu **UTC**.

Selain menggunakan `time.UTC` untuk penentuan lokasi, tersedia juga `time.Local` yang nilainya adalah relatif terhadap waktu lokal kita.

## A.8.2. Method Milik `time.Time`

Tipe data `time.Time` merupakan struct, memiliki beberapa method yang bisa dipakai.

```
var now = time.Now()
fmt.Println("year:", now.Year(), "month:", now.Month())
// year: 2015 month: 8
```

Kode di atas adalah contoh penggunaan beberapa method milik objek bertipe `time.Time`. Method `Year()` mengembalikan informasi tahun, dan `Month()` mengembalikan informasi angka bulan.

Selain kedua method di atas, ada banyak lagi yang bisa dimanfaatkan. Tabel berikut merupakan list method yang berhubungan dengan `date`, `time`, dan `location` yang dimiliki tipe `time.Time`.

Method	Return Type	Penjelasan
<code>now.Year()</code>	<code>int</code>	Tahun
<code>now.YearDay()</code>	<code>int</code>	Hari ke-? di mulai awal tahun
<code>now.Month()</code>	<code>int</code>	Bulan
<code>now.Weekday()</code>	<code>string</code>	Nama hari. Bisa menggunakan <code>now.Weekday().String()</code> untuk mengambil bentuk string-nya
<code>now.ISOWeek()</code>	<code>( int , int )</code>	Tahun dan minggu ke-? mulai awal tahun
<code>now.Day()</code>	<code>int</code>	Tanggal
<code>now.Hour()</code>	<code>int</code>	Jam
<code>now.Minute()</code>	<code>int</code>	Menit
<code>now.Second()</code>	<code>int</code>	Detik
<code>now.Nanosecond()</code>	<code>int</code>	Nano detik
<code>now.Local()</code>	<code>time.Time</code>	Waktu dalam timezone lokal
<code>now.Location()</code>	<code>*time.Location</code>	Mengambil informasi lokasi, apakah <i>local</i> atau <i>utc</i> . Bisa menggunakan <code>now.Location().String()</code> untuk mengambil bentuk string-nya
<code>now.Zone()</code>	<code>( string , int )</code>	Mengembalikan informasi <i>timezone offset</i> dalam string dan numerik. Sebagai contoh <code>WIB, 25200</code>
<code>now.IsZero()</code>	<code>bool</code>	Deteksi apakah nilai object <code>now</code> adalah <code>01 Januari tahun 1, 00:00:00 UTC</code> . Jika iya maka bernilai <code>true</code>
<code>now.UTC()</code>	<code>time.Time</code>	Waktu dalam timezone <code>UTC</code>
<code>now.Unix()</code>	<code>int64</code>	Waktu dalam format <i>unix time</i>
<code>now.UnixNano()</code>	<code>int64</code>	Waktu dalam format <i>unix time</i> . Infomasi nano detik juga dimasukkan
<code>now.String()</code>	<code>string</code>	Waktu dalam string

## A.8.3. Parsing `time.Time`

Data `string` bisa dikonversi menjadi `time.Time` dengan memanfaatkan `time.Parse`. Fungsi ini membutuhkan 2 parameter:

- Parameter ke-1 adalah layout format dari data waktu yang akan diparsing.
- Parameter ke-2 adalah data string yang ingin diparsing.

Contoh penerapannya bisa dilihat di kode berikut.

```
var layoutFormat, value string
var date time.Time

layoutFormat = "2006-01-02 15:04:05"
value = "2015-09-02 08:04:00"
date, _ = time.Parse(layoutFormat, value)
fmt.Println(value, "\t->", date.String())
// 2015-09-02 08:04:00 +0000 UTC

layoutFormat = "02/01/2006 MST"
value = "02/09/2015 WIB"
date, _ = time.Parse(layoutFormat, value)
fmt.Println(value, "\t\t->", date.String())
// 2015-09-02 00:00:00 +0700 WIB
```

```
[novalagung:belajar-golang $ go run bab38.go
2015-09-02 08:04:00      -> 2015-09-02 08:04:00 +0000 UTC
02/09/2015 WIB          -> 2015-09-02 00:00:00 +0700 WIB
novalagung:belajar-golang $ ]
```

Layout format time di golang berbeda dibanding bahasa lain. Umumnya layout format yang digunakan adalah seperti "DD/MM/YYYY" , di Golang tidak.

Golang memiliki standar layout format yang cukup unik, contohnya seperti pada kode di atas "2006-01-02 15:04:05" . Golang menggunakan `2006` untuk parsing tahun, bukan `YYYY` ; `01` untuk parsing bulan; `02` untuk parsing hari; dan seterusnya. Detailnya bisa dilihat di tabel berikut.

Layout Format	Penjelasan	Contoh Data
<code>2006</code>	Tahun 4 digit	<code>2015</code>
<code>006</code>	Tahun 3 digit	<code>015</code>
<code>06</code>	Tahun 2 digit	<code>15</code>
<code>01</code>	Bulan 2 digit	<code>05</code>
<code>1</code>	Bulan 1 digit jika dibawah bulan 10, selainnya 2 digit	<code>5 , 12</code>
<code>January</code>	Nama bulan dalam bahasa inggris	<code>September , August</code>
<code>Jan</code>	Nama bulan dalam bahasa inggris, 3 huruf	<code>Sep , Aug</code>
<code>02</code>	Tanggal 2 digit	<code>02</code>
<code>2</code>	Tanggal 1 digit jika dibawah bulan 10, selainnya 2 digit	<code>8 , 31</code>
<code>Monday</code>	Nama hari dalam bahasa inggris	<code>Saturday , Friday</code>
<code>Mon</code>	Nama hari dalam bahasa inggris, 3 huruf	<code>Sat , Fri</code>
<code>15</code>	Jam dengan format <b>24 jam</b>	<code>18</code>
<code>03</code>	Jam dengan format <b>12 jam</b> 2 digit	<code>05 , 11</code>
<code>3</code>	Jam dengan format <b>12 jam</b> 1 digit jika dibawah jam 11, selainnya 2 digit	<code>5 , 11</code>
<code>PM</code>	AM/PM, biasa digunakan dengan format jam <b>12 jam</b>	<code>PM , AM</code>
<code>04</code>	Menit 2 digit	<code>08</code>

4	Menit 1 digit jika dibawah menit 10, selainnya 2 digit	8 , 24
05	Detik 2 digit	06
5	Detik 1 digit jika dibawah detik 10, selainnya 2 digit	6 , 36
999999	Nano detik	124006
MST	Lokasi timezone	UTC , WIB , EST
Z0700	Offset timezone	Z , +0700 , -0200

## A.8.4. Predefined Layout Format Untuk Keperluan Parsing Time

Golang juga menyediakan beberapa predefined layout format umum yang bisa dimanfaatkan. Jadi tidak perlu menuliskan kombinasi komponen-komponen layout format.

Salah satu predefined layout yang bisa digunakan adalah `time.RFC822`. Format ini sama dengan `02 Jan 06 15:04 MST`. Berikut adalah contoh penerapannya.

```
var date, _ = time.Parse(time.RFC822, "02 Sep 15 08:00 WIB")
fmt.Println(date.String())
// 2015-09-02 08:00:00 +0700 WIB
```

Ada beberapa layout format lain yang tersedia, silakan lihat tabel berikut.

Predefined Layout Format	Layout Format
<code>time.ANSIC</code>	Mon Jan _2 15:04:05 2006
<code>time.UnixDate</code>	Mon Jan _2 15:04:05 MST 2006
<code>time.RubyDate</code>	Mon Jan 02 15:04:05 -0700 2006
<code>time.RFC822</code>	02 Jan 06 15:04 MST
<code>time.RFC822Z</code>	02 Jan 06 15:04 -0700
<code>time.RFC850</code>	Monday, 02-Jan-06 15:04:05 MST
<code>time.RFC1123</code>	Mon, 02 Jan 2006 15:04:05 MST
<code>time.RFC1123Z</code>	Mon, 02 Jan 2006 15:04:05 -0700
<code>time.RFC3339</code>	2006-01-02T15:04:05Z07:00
<code>time.RFC3339Nano</code>	2006-01-02T15:04:05.999999999Z07:00
<code>time.Kitchen</code>	3:04PM
<code>time.Stamp</code>	Jan _2 15:04:05
<code>time.StampMilli</code>	Jan _2 15:04:05.000
<code>time.StampMicro</code>	Jan _2 15:04:05.000000
<code>time.StampNano</code>	Jan _2 15:04:05.000000000

## A.8.5. Format `time.Time`

Setelah sebelumnya kita belajar tentang cara konversi data dengan tipe `string` ke `time.Time`. Kali ini kita akan belajar kebalikannya, konversi `time.Time` ke `string`.

Method `Format()` milik tipe `time.Time` digunakan untuk membentuk output `string` sesuai dengan layout format yang diinginkan. Contoh bisa dilihat pada kode berikut.

```
var date, _ = time.Parse(time.RFC822, "02 Sep 15 08:00 WIB")
var dateS1 = date.Format("Monday 02, January 2006 15:04 MST")
fmt.Println("dateS1", dateS1)
// Wednesday 02, September 2015 08:00 WIB

var dateS2 = date.Format(time.RFC3339)
fmt.Println("dateS2", dateS2)
// 2015-09-02T08:00:00+07:00
```

Variabel `date` di atas berisikan hasil parsing data dengan format `time.RFC822`. Data tersebut kemudian diformat sebagai string 2 kali dengan layout format berbeda.

```
[novalagung:belajar-golang $ go run bab38.go
dateS1 Wednesday 02, September 2015 08:00 WIB
dateS2 2015-09-02T08:00:00+07:00
novalagung:belajar-golang $ ]
```

## A.8.6. Handle Error Parsing `time.Time`

Ketika parsing `string` ke `time.Time`, sangat memungkinkan bisa terjadi error karena struktur data yang akan diparsing tidak sesuai layout format yang digunakan.

Error tidaknya parsing bisa diketahui lewat nilai kembalian ke-2 fungsi `time.Parse`. Berikut adalah contoh penerapannya.

```
var date, err = time.Parse("06 Jan 15", "02 Sep 15 08:00 WIB")

if err != nil {
    fmt.Println("error", err.Error())
    return
}

fmt.Println(date)
```

Kode di atas menghasilkan error karena format tidak sesuai dengan skema data yang akan diparsing. Layout format yang seharusnya digunakan adalah `06 Jan 15 03:04 MST`.

```
[novalagung:belajar-golang $ go run bab38.go
error parsing time "02 Sep 15 08:00 WIB": extra text: 08:00 WIB
novalagung:belajar-golang $ ]
```

## A.39. Timer

Ada beberapa fungsi dalam package `time` yang bisa dimanfaatkan untuk menunda atau mengatur jadwal eksekusi sebuah proses dalam jeda waktu tertentu.

### A.39.1. Fungsi `time.Sleep()`

Fungsi ini digunakan untuk menghentikan program sejenak. `time.Sleep()` bersifat **blocking**, statement dibawahnya tidak akan dieksekusi sampai waktu pemberhentian usai. Contoh sederhana penerapan bisa dilihat pada kode berikut.

```
package main

import "fmt"
import "time"

func main () {
    fmt.Println("start")
    time.Sleep(time.Second * 4)
    fmt.Println("after 4 seconds")
}
```

Hasilnya, tulisan `"start"` muncul, lalu 4 detik kemudian tulisan `"after 4 seconds"` muncul.

### A.39.2. Fungsi `time.NewTimer()`

Fungsi ini sedikit berbeda dengan `time.Sleep()`. Fungsi `time.NewTimer()` mengembalikan objek bertipe `*time.Timer` yang memiliki property `c`. Cara kerja fungsi ini, setelah jeda waktu yang ditentukan sebuah data akan dikirimkan lewat channel `c`. Penggunaan fungsi ini harus diikuti dengan statement untuk penerimaan data dari channel `c`.

Untuk lebih jelasnya silakan perhatikan kode berikut.

```
var timer = time.NewTimer(4 * time.Second)
fmt.Println("start")
<-timer.c
fmt.Println("finish")
```

Statement `var timer = time.NewTimer(4 * time.Second)` mengindikasikan bahwa nantinya akan ada data yang dikirimkan ke channel `timer.c` setelah 4 detik berlalu. Baris kode `<-timer.c` menandakan penerimaan data dari channel `timer.c`. Karena penerimaan channel sendiri sifatnya adalah blocking, maka statement `fmt.Println("finish")` baru akan dieksekusi setelah **4 detik**.

Hasil program di atas adalah tulisan `"start"` muncul, lalu setelah 4 detik tulisan `"expired"` muncul.

### A.39.3. Fungsi `time.AfterFunc()`

Fungsi `time.AfterFunc()` memiliki 2 parameter. Parameter pertama adalah durasi timer, dan parameter kedua adalah *callback* nya. Callback tersebut akan dieksekusi jika waktu sudah memenuhi durasi timer.

```
var ch = make(chan bool)
```

```

time.AfterFunc(4*time.Second, func() {
    fmt.Println("expired")
    ch <- true
})

fmt.Println("start")
<-ch
fmt.Println("finish")

```

Hasil dari kode di atas, tulisan "start" muncul kemudian setelah 4 detik berlalu, tulisan "expired" muncul.

Dalam callback terdapat proses transfer data lewat channel, menjadikan tulisan "finish" akan muncul tepat setelah tulisan "expired" muncul.

Beberapa hal yang perlu diketahui dalam menggunakan fungsi ini:

- Jika tidak ada serah terima data lewat channel, maka eksekusi `time.AfterFunc()` adalah asynchronous dan tidak blocking.
- Jika ada serah terima data lewat channel, maka fungsi akan tetap berjalan asynchronous dan tidak blocking hingga baris kode dimana penerimaan data channel dilakukan.

## A.39.4. Fungsi `time.After()`

Kegunaan fungsi ini mirip seperti `time.Sleep()`. Perbedaannya adalah, fungsi `timer.After()` akan mengembalikan data channel, sehingga perlu menggunakan tanda `<-` dalam penerapannya.

```

<-time.After(4 * time.Second)
fmt.Println("expired")

```

Tulisan "expired" akan muncul setelah 4 detik.

## A.39.5. Kombinasi Timer & Goroutine

Berikut merupakan contoh penerapan timer dan goroutine. Program di bawah ini adalah program tanya-jawab sederhana. Sebuah pertanyaan muncul dan user harus menginputkan jawaban dalam waktu tidak lebih dari 5 detik. Jika 5 detik berlalu dan belum ada jawaban, maka akan muncul pesan **time out**.

OK langsung saja, mari kita buat programnya, pertama, import package yang diperlukan.

```

package main

import "fmt"
import "os"
import "time"

```

Buat fungsi `timer()`, nantinya fungsi ini dieksekusi sebagai goroutine. Di dalam fungsi `timer()` terdapat blok kode jika waktu sudah mencapai `timeout`, maka sebuah data dikirimkan lewat channel `ch`.

```

func timer(timeout int, ch chan<- bool) {
    time.AfterFunc(time.Duration(timeout)*time.Second, func() {
        ch <- true
    })
}

```

Siapkan juga fungsi `watcher()`. Fungsi ini juga akan dieksekusi sebagai goroutine. Tugasnya cukup sederhana, yaitu menerima data dari channel `ch` (jika ada penerimaan data, berarti sudah masuk waktu timeout), lalu menampilkan pesan bahwa waktu telah habis.

```
func watcher(timeout int, ch <-chan bool) {
    <-ch
    fmt.Println("\ntime out! no answer more than", timeout, "seconds")
    os.Exit(0)
}
```

Terakhir, buat implementasi di fungsi `main`.

```
func main() {
    var timeout = 5
    var ch = make(chan bool)

    go timer(timeout, ch)
    go watcher(timeout, ch)

    var input string
    fmt.Print("what is 725/25 ? ")
    fmt.Scan(&input)

    if input == "29" {
        fmt.Println("the answer is right!")
    } else {
        fmt.Println("the answer is wrong!")
    }
}
```

Ketika user tidak menginputkan apa-apa dalam kurun waktu 5 detik, maka akan muncul pesan timeout, lalu program dihentikan.

```
[novalagung:belajar-golang $ go run bab39.go
what is 725/25 ? 34
the answer is wrong!
[novalagung:belajar-golang $ go run bab39.go
what is 725/25 ?
time out! no answer more than 5 seconds
[novalagung:belajar-golang $ go run bab39.go
what is 725/25 ? 29
the answer is right!
novalagung:belajar-golang $ ]
```

## A.40. Konversi Antar Tipe Data

Di bab-bab sebelumnya kita sudah mengaplikasikan beberapa cara konversi data, contohnya seperti konversi `string` ↔ `int` menggunakan `strconv`, dan `time.Time` ↔ `string`. Di bab ini kita akan belajar lebih banyak.

### A.40.1. Konversi Menggunakan `strconv`

Package `strconv` berisi banyak fungsi yang sangat membantu kita untuk melakukan konversi. Berikut merupakan beberapa fungsi yang dalam package tersebut.

#### A.40.1.a. Fungsi `strconv.Atoi()`

Fungsi ini digunakan untuk konversi data dari tipe `string` ke `int`. `strconv.Atoi()` menghasilkan 2 buah nilai kembalian, yaitu hasil konversi dan `error` (jika konversi sukses, maka `error` berisi `nil`).

```
package main

import "fmt"
import "strconv"

func main() {
    var str = "124"
    var num, err = strconv.Atoi(str)

    if err == nil {
        fmt.Println(num) // 124
    }
}
```

#### A.40.1.b. Fungsi `strconv.Itoa()`

Merupakan kebalikan dari `strconv.Atoi`, berguna untuk konversi `int` ke `string`.

```
var num = 124
var str = strconv.Itoa(num)

fmt.Println(str) // "124"
```

#### A.40.1.c. Fungsi `strconv.ParseInt()`

Digunakan untuk konversi `string` berbentuk numerik dengan basis tertentu ke tipe numerik non-desimal dengan lebar data bisa ditentukan.

Pada contoh berikut, string `"124"` dikonversi ke tipe numerik dengan ketentuan basis yang digunakan `10` dan lebar datanya mengikuti tipe `int64` (lihat parameter ketiga).

```
var str = "124"
var num, err = strconv.ParseInt(str, 10, 64)

if err == nil {
    fmt.Println(num) // 124
}
```

Contoh lainnya, string "1010" dikonversi ke basis 2 (biner) dengan tipe data hasil adalah `int8`.

```
var str = "1010"
var num, err = strconv.ParseInt(str, 2, 8)

if err == nil {
    fmt.Println(num) // 10
}
```

### A.40.1.d. Fungsi `strconv.FormatInt()`

Berguna untuk konversi data numerik `int64` ke `string` dengan basis numerik bisa ditentukan sendiri.

```
var num = int64(24)
var str = strconv.FormatInt(num, 8)

fmt.Println(str) // 30
```

### A.40.1.e. Fungsi `strconv.ParseFloat()`

Digunakan untuk konversi `string` ke numerik desimal dengan lebar data bisa ditentukan.

```
var str = "24.12"
var num, err = strconv.ParseFloat(str, 32)

if err == nil {
    fmt.Println(num) // 24.1200008392334
}
```

Pada contoh di atas, string "24.12" dikonversi ke float dengan lebar tipe data `float32`. Hasil konversi `strconv.ParseFloat` adalah sesuai dengan standar [IEEE Standard for Floating-Point Arithmetic](#).

### A.40.1.f. Fungsi `strconv.FormatFloat()`

Berguna untuk konversi data bertipe `float64` ke `string` dengan format eksponen, lebar digit desimal, dan lebar tipe data bisa ditentukan.

```
var num = float64(24.12)
var str = strconv.FormatFloat(num, 'f', 6, 64)

fmt.Println(str) // 24.120000
```

Pada kode di atas, Data `24.12` yang bertipe `float64` dikonversi ke string dengan format eksponen `f` atau tanpa eksponen, lebar digit desimal 6 digit, dan lebar tipe data `float64`.

Ada beberapa format eksponen yang bisa digunakan. Detailnya bisa dilihat di tabel berikut.

Format Eksponen	Penjelasan
b	-ddddp±ddd, a, eksponen biner (basis 2)
e	-d.ddde±dd, a, eksponen desimal (basis 10)
E	-d.ddddE±dd, a, eksponen desimal (basis 10)
f	-ddd.dddd, tanpa eksponen
g	Akan menggunakan format eksponen e untuk eksponen besar dan f untuk selainnya

G

Akan menggunakan format eksponen `E` untuk eksponen besar dan `f` untuk selainnya

### A.40.1.g. Fungsi `strconv.ParseBool()`

Digunakan untuk konversi `string` ke `bool`.

```
var str = "true"
var bul, err = strconv.ParseBool(str)

if err == nil {
    fmt.Println(bul) // true
}
```

### A.40.1.h. Fungsi `strconv.FormatBool()`

Digunakan untuk konversi `bool` ke `string`.

```
var bul = true
var str = strconv.FormatBool(bul)

fmt.Println(str) // 124
```

## A.40.2. Konversi Data Menggunakan Casting

Keyword tipe data bisa digunakan untuk casting. Cara penggunaannya adalah dengan menuliskan tipe data sebagai fungsi dan menyisipkan data yang akan dikonversi sebagai parameternya.

```
var a float64 = float64(24)
fmt.Println(a) // 24

var b int32 = int32(24.00)
fmt.Println(b) // 24
```

### A.40.3. Casting `string` ↔ `byte`

String sebenarnya adalah slice/array `byte`. Di GoLang sebuah karakter biasa (bukan unicode) direpresentasikan oleh sebuah elemen slice byte. Tiap elemen slice berisi data `int` dengan basis desimal, yang merupakan kode ASCII dari karakter dalam string.

Cara mendapatkan slice byte dari sebuah data string adalah dengan meng-casting-nya ke tipe `[]byte`.

```
var text1 = "halo"
var b = []byte(text1)

fmt.Printf("%d %d %d %d \n", b[0], b[1], b[2], b[3])
// 104 97 108 111
```

Pada contoh di atas, string dalam variabel `text1` dikonversi ke `[]byte`. Tiap elemen slice byte tersebut kemudian ditampilkan satu-per-satu.

Contoh berikut ini merupakan kebalikan dari contoh di atas, data bertipe `[]byte` akan dicari bentuk `string`-nya.

```
var byte1 = []byte{104, 97, 108, 111}
var s = string(byte1)
```

```
fmt.Printf("%s \n", s)
// halo
```

Pada contoh di atas, beberapa kode byte dituliskan dalam bentuk slice, ditampung variabel `byte1`. Lalu, nilai variabel tersebut di-cast ke `string`, untuk kemudian ditampilkan.

Selain itu, tiap karakter string juga bisa di-casting ke bentuk `int`, hasilnya adalah sama yaitu data byte dalam bentuk numerik basis desimal, dengan ketentuan literal string yang digunakan adalah tanda petik satu (`'`).

Juga berlaku sebaliknya, data numerik jika di-casting ke bentuk string dideteksi sebagai kode ASCII dari karakter yang akan dihasilkan.

```
var c int64 = int64('h')
fmt.Println(c) // 104

var d string = string(104)
fmt.Println(d) // h
```

## A.40.4. Konversi Data `interface{}` Menggunakan Teknik Type Assertions

**Type assertions** merupakan teknik casting data `interface{}` ke segala jenis tipe (dengan syarat data tersebut memang bisa di-casting ke tipe tujuan).

Berikut merupakan contoh penerapannya. Variabel `data` disiapkan bertipe `map[string]interface{}`, berisikan beberapa item dengan tipe data value nya berbeda satu sama lain.

```
var data = map[string]interface{}{
    "nama":      "john wick",
    "grade":     2,
    "height":    156.5,
    "isMale":    true,
    "hobbies":   []string{"eating", "sleeping"},
}

fmt.Println(data["nama"].(string))
fmt.Println(data["grade"].(int))
fmt.Println(data["height"].(float64))
fmt.Println(data["isMale"].(bool))
fmt.Println(data["hobbies"].([]string))
```

Statement `data["nama"].(string)` maksudnya adalah, nilai `data["nama"]` dicasting sebagai `string`.

Tipe asli data pada variabel `interface{}` bisa diketahui dengan cara meng-casting ke tipe `type`. Namun casting ke tipe `type` hanya bisa dilakukan pada `switch`.

```
for _, val := range data {
    switch val.(type) {
    case string:
        fmt.Println(val.(string))
    case int:
        fmt.Println(val.(int))
    case float64:
        fmt.Println(val.(float64))
    case bool:
        fmt.Println(val.(bool))
    case []string:
        fmt.Println(val.([]string))
```

```
    default:
        fmt.Println(val.(int))
    }
}
```

Kombinasi `switch - case` bisa dimanfaatkan untuk deteksi tipe asli sebuah data bertipe `interface{}`, contoh penerapannya seperti pada kode di atas.

## A.41. Fungsi String

Golang menyediakan package `strings`, berisikan cukup banyak fungsi untuk keperluan pengolahan data string. Bab ini berisi pembahasan mengenai beberapa fungsi yang ada di dalam package tersebut.

### A.41.1. Fungsi `strings.Contains()`

Dipakai untuk deteksi apakah string (parameter kedua) merupakan bagian dari string lain (parameter pertama). Nilai kembalinya berupa `bool`.

```
package main

import "fmt"
import "strings"

func main() {
    var exists = strings.Contains("john wick", "wick")
    fmt.Println(exists)
}
```

Variabel `exists` akan bernilai `true`, karena string `"wick"` merupakan bagian dari `"john wick"`.

### A.41.2. Fungsi `strings.HasPrefix()`

Digunakan untuk deteksi apakah sebuah string (parameter pertama) diawali string tertentu (parameter kedua).

```
var isPrefix1 = strings.HasPrefix("john wick", "jo")
fmt.Println(isPrefix1) // true

var isPrefix2 = strings.HasPrefix("john wick", "wi")
fmt.Println(isPrefix2) // false
```

### A.41.3. Fungsi `strings.HasSuffix()`

Digunakan untuk deteksi apakah sebuah string (parameter pertama) diakhiri string tertentu (parameter kedua).

```
var isSuffix1 = strings.HasSuffix("john wick", "ic")
fmt.Println(isSuffix1) // false

var isSuffix2 = strings.HasSuffix("john wick", "ck")
fmt.Println(isSuffix2) // true
```

### A.41.4. Fungsi `strings.Count()`

Memiliki kegunaan untuk menghitung jumlah karakter tertentu (parameter kedua) dari sebuah string (parameter pertama). Nilai kembalian fungsi ini adalah jumlah karakternya.

```
var howMany = strings.Count("ethan hunt", "t")
fmt.Println(howMany) // 2
```

Nilai yang dikembalikan `2`, karena pada string `"ethan hunt"` terdapat dua buah karakter `"t"`.

## A.41.5. Fungsi `strings.Index()`

Digunakan untuk mencari posisi indeks sebuah string (parameter kedua) dalam string (parameter pertama).

```
var index1 = strings.Index("ethan hunt", "ha")
fmt.Println(index1) // 2
```

String `"ha"` berada pada posisi ke `2` dalam string `"ethan hunt"` (indeks dimulai dari 0). Jika diketemukan dua substring, maka yang diambil adalah yang pertama, contoh:

```
var index2 = strings.Index("ethan hunt", "n")
fmt.Println(index2) // 4
```

String `"n"` berada pada indeks `4` dan `8`. Yang dikembalikan adalah yang paling kiri (paling kecil), yaitu `4`.

## A.41.6. Fungsi `strings.Replace()`

Fungsi ini digunakan untuk replace atau mengganti bagian dari string dengan string tertentu. Jumlah substring yang di-replace bisa ditentukan, apakah hanya 1 string pertama, 2 string, atau kesemuanya.

```
var text = "banana"
var find = "a"
var replaceWith = "o"

var newText1 = strings.Replace(text, find, replaceWith, 1)
fmt.Println(newText1) // "bonana"

var newText2 = strings.Replace(text, find, replaceWith, 2)
fmt.Println(newText2) // "bonona"

var newText3 = strings.Replace(text, find, replaceWith, -1)
fmt.Println(newText3) // "bonono"
```

Penjelasan:

- Pada contoh di atas, substring `"a"` pada string `"banana"` akan di-replace dengan string `"o"`.
- Pada `newText1`, hanya 1 huruf `o` saja yang tereplace karena maksimal substring yang ingin di-replace ditentukan 1.
- Angka `-1` akan menjadikan proses replace berlaku pada semua substring. Contoh bisa dilihat pada `newText3`.

## A.41.7. Fungsi `strings.Repeat()`

Digunakan untuk mengulang string (parameter pertama) sebanyak data yang ditentukan (parameter kedua).

```
var str = strings.Repeat("na", 4)
fmt.Println(str) // "nananana"
```

Pada contoh di atas, string `"na"` diulang sebanyak 4 kali. Hasilnya adalah: `"nananana"`

## A.41.8. Fungsi `strings.Split()`

Digunakan untuk memisah string (parameter pertama) dengan tanda pemisah bisa ditentukan sendiri (parameter kedua). Hasilnya berupa array string.

```
var string1 = strings.Split("the dark knight", " ")
fmt.Println(string1) // ["the", "dark", "knight"]

var string2 = strings.Split("batman", "")
fmt.Println(string2) // ["b", "a", "t", "m", "a", "n"]
```

String `"the dark knight"` dipisah oleh karakter spasi `" "`, hasilnya kemudian ditampung oleh `string1`.

Untuk memisah string menjadi array tiap 1 string, gunakan pemisah string kosong `""`. Bisa dilihat contohnya pada variabel `string2`.

## A.41.9. Fungsi `strings.Join()`

Memiliki kegunaan berkebalikan dengan `strings.Split()`. Digunakan untuk menggabungkan array string (parameter pertama) menjadi sebuah string dengan pemisah tertentu (parameter kedua).

```
var data = []string{"banana", "papaya", "tomato"}
var str = strings.Join(data, "-")
fmt.Println(str) // "banana-papaya-tomato"
```

Array `data` digabungkan menjadi satu dengan pemisah tanda *dash* (`-`).

## A.41.10. Fungsi `strings.ToLower()`

Mengubah huruf-huruf string menjadi huruf kecil.

```
var str = strings.ToLower("aLay")
fmt.Println(str) // "alay"
```

## A.41.11. Fungsi `strings.ToUpper()`

Mengubah huruf-huruf string menjadi huruf besar.

```
var str = strings.ToUpper("eat!")
fmt.Println(str) // "EAT!"
```

## A.42. Regex

Regex atau **regular expression** adalah suatu teknik yang digunakan untuk pencocokan string dengan pola tertentu. Regex biasa dimanfaatkan untuk pencarian dan pengubahan data string.

Golang mengadopsi standar regex **RE2**, untuk melihat sintaks yang di-support engine ini bisa langsung merujuk ke dokumentasinya di <https://github.com/google/re2/wiki/Syntax>.

Pada bab ini kita akan belajar mengenai pengaplikasian regex dengan memanfaatkan fungsi-fungsi dalam package `regexp`.

### A.42.1. Penerapan Regexp

Fungsi `regexp.Compile()` digunakan untuk mengkompilasi ekspresi regex. Fungsi tersebut mengembalikan objek bertipe `regexp.*Regexp`.

Berikut merupakan contoh penerapan regex untuk pencarian karakter.

```
package main

import "fmt"
import "regexp"

func main() {
    var text = "banana burger soup"
    var regex, err = regexp.Compile(`[a-z]+`)

    if err != nil {
        fmt.Println(err.Error())
    }

    var res1 = regex.FindAllString(text, 2)
    fmt.Printf("%#v \n", res1)
    // ["banana", "burger"]

    var res2 = regex.FindAllString(text, -1)
    fmt.Printf("%#v \n", res2)
    // ["banana", "burger", "soup"]
}
```

Ekspresi `[a-z]+` maknanya adalah, semua string yang merupakan alphabet yang hurufnya kecil. Ekspresi tersebut di-compile oleh `regexp.Compile()` lalu disimpan ke variabel objek `regex` bertipe `regexp.*Regexp`.

Struct `regexp.Regexp` memiliki banyak method, salah satunya adalah `FindAllString()`, berfungsi untuk mencari semua string yang sesuai dengan ekspresi regex, dengan kembalian berupa array string.

Jumlah hasil pencarian dari `regex.FindAllString()` bisa ditentukan. Contohnya pada `res1`, ditentukan maksimal `2` data saja pada nilai kembalian. Jika batas di set `-1`, maka akan mengembalikan semua data.

Ada cukup banyak method struct `regexp.*Regexp` yang bisa kita manfaatkan untuk keperluan pengelolaan string. Berikut merupakan pembahasan tiap method-nya.

### A.42.2. Method `MatchString()`

Method ini digunakan untuk mendeteksi apakah string memenuhi sebuah pola regexp.

```
var text = "banana burger soup"
var regex, _ = regexp.MustCompile(`[a-z]+`)

var isMatch = regex.MatchString(text)
fmt.Println(isMatch)
// true
```

Pada contoh di atas `isMatch` bernilai `true` karena string "banana burger soup" memenuhi pola regex `[a-z]+`.

### A.42.3. Method `FindString()`

Digunakan untuk mencari string yang memenuhi kriteria regexp yang telah ditentukan.

```
var text = "banana burger soup"
var regex, _ = regexp.MustCompile(`[a-z]+`)

var str = regex.FindString(text)
fmt.Println(str)
// "banana"
```

Fungsi ini hanya mengembalikan 1 buah hasil saja. Jika ada banyak substring yang sesuai dengan ekspresi regexp, akan dikembalikan yang pertama saja.

### A.42.4. Method `FindStringIndex()`

Digunakan untuk mencari index string kembalian hasil dari operasi regexp.

```
var text = "banana burger soup"
var regex, _ = regexp.MustCompile(`[a-z]+`)

var idx = regex.FindStringIndex(text)
fmt.Println(idx)
// [0, 6]

var str = text[0:6]
fmt.Println(str)
// "banana"
```

Method ini sama dengan `FindString()` hanya saja yang dikembalikan indeks-nya.

### A.42.5. Method `FindAllString()`

Digunakan untuk mencari banyak string yang memenuhi kriteria regexp yang telah ditentukan.

```
var text = "banana burger soup"
var regex, _ = regexp.MustCompile(`[a-z]+`)

var str1 = regex.FindAllString(text, -1)
fmt.Println(str1)
// ["banana", "burger", "soup"]

var str2 = regex.FindAllString(text, 1)
fmt.Println(str2)
// ["banana"]
```

Jumlah data yang dikembalikan bisa ditentukan. Jika diisi dengan `-1`, maka akan mengembalikan semua data.

## A.42.6. Method ReplaceAllString()

Berguna untuk me-replace semua string yang memenuhi kriteria regexp, dengan string lain.

```
var text = "banana burger soup"
var regex, _ = regexp.MustCompile(`[a-z]+`)

var str = regex.ReplaceAllString(text, "potato")
fmt.Println(str)
// "potato potato potato"
```

## A.42.7. Method ReplaceAllStringFunc()

Digunakan untuk me-replace semua string yang memenuhi kriteria regexp, dengan kondisi yang bisa ditentukan untuk setiap substring yang akan di replace.

```
var text = "banana burger soup"
var regex, _ = regexp.MustCompile(`[a-z]+`)

var str = regex.ReplaceAllStringFunc(text, func(each string) string {
    if each == "burger" {
        return "potato"
    }
    return each
})
fmt.Println(str)
// "banana potato soup"
```

Pada contoh di atas, jika salah satu substring yang *match* adalah `"burger"` maka akan diganti dengan `"potato"`, string selainnya tidak di replace.

## A.42.8. Method Split()

Digunakan untuk memisah string dengan pemisah adalah substring yang memenuhi kriteria regexp yang telah ditentukan.

Jumlah karakter yang akan di split bisa ditentukan dengan mengisi parameter kedua fungsi `regex.Split()`. Jika di-isi `-1` maka semua karakter yang memenuhi regex akan di-replace. Contoh lain, jika di-isi `2`, maka hanya 2 karakter pertama yang memenuhi regex akan di-replace.

```
var text = "banana,burger,soup"
var regex, _ = regexp.MustCompile(`[a-z]+`)

var str = regex.Split(text, -1)
fmt.Printf("%#v \n", str)
// [ "", "", "", "", "" ]
```

## A.43. Encode - Decode Base64

Golang memiliki package `encoding/base64`, berisikan fungsi-fungsi untuk kebutuhan **encode** dan **decode** data ke base64 dan sebaliknya. Data yang akan di-encode harus bertipe `[]byte`, perlu dilakukan casting untuk data-data yang belum sesuai tipenya.

Ada beberapa cara yang bisa digunakan untuk encode dan decode data, dan di bab ini kita akan mempelajarinya.

### A.43.1. Penerapan Fungsi `EncodeToString()` & `DecodeString()`

Fungsi `EncodeToString()` digunakan untuk encode data dari bentuk string ke base46. Fungsi `DecodeString()` melakukan kebalikan dari `EncodeToString()`. Berikut adalah contoh penerapannya.

```
package main

import "encoding/base64"
import "fmt"

func main() {
    var data = "john wick"

    var encodedString = base64.StdEncoding.EncodeToString([]byte(data))
    fmt.Println("encoded:", encodedString)

    var decodedByte, _ = base64.StdEncoding.DecodeString(encodedString)
    var decodedString = string(decodedByte)
    fmt.Println("decoded:", decodedString)
}
```

Variabel `data` yang bertipe `string`, harus di-casting terlebih dahulu kedalam bentuk `[]byte` sebelum di-encode menggunakan fungsi `base64.StdEncoding.EncodeToString()`. Hasil encode adalah data base64 bertipe `string`.

Sedangkan pada fungsi decode `base64.StdEncoding.DecodeString()`, data base64 bertipe `string` di-decode kembali ke string aslinya, tapi bertipe `[]byte`. Ekspresi `string(decodedByte)` menjadikan data `[]byte` tersebut berubah menjadi string.

```
[novalagung:belajar-golang $ go run bab43.go
encoded: am9obiB3aWNr
decoded: john wick
novalagung:belajar-golang $ ]
```

### A.43.2. Penerapan Fungsi `Encode()` & `Decode()`

Kedua fungsi ini kegunaannya sama dengan fungsi yang sebelumnya kita bahas, salah satu pembedanya adalah data yang akan dikonversi dan hasilnya bertipe `[]byte`. Penggunaan cara ini cukup panjang karena variabel penyimpan hasil encode maupun decode harus disiapkan terlebih dahulu, dan harus memiliki lebar data sesuai dengan hasil yang akan ditampung (yang nilainya bisa dicari menggunakan fungsi `EncodedLen()` dan `DecodedLen()`).

Lebih jelasnya silakan perhatikan contoh berikut.

```
var data = "john wick"

var encoded = make([]byte, base64.StdEncoding.EncodedLen(len(data)))
base64.StdEncoding.Encode(encoded, []byte(data))
```

```

var encodedString = string(encoded)
fmt.Println(encodedString)

var decoded = make([]byte, base64.StdEncoding.DecodedLen(len(encoded)))
var _, err = base64.StdEncoding.Decode(decoded, encoded)
if err != nil {
    fmt.Println(err.Error())
}
var decodedString = string(decoded)
fmt.Println(decodedString)

```

Fungsi `base64.StdEncoding.EncodedLen(len(data))` menghasilkan informasi lebar data-ketika-sudah-di-encode. Nilai tersebut kemudian ditentukan sebagai lebar alokasi tipe `[]byte` pada variabel `decoded` yang nantinya digunakan untuk menampung hasil encoding.

Fungsi `base64.StdEncoding.DecodedLen()` memiliki kegunaan sama dengan `EncodedLen()`, hanya saja digunakan untuk keperluan decoding.

Dibanding 2 fungsi sebelumnya, fungsi `Encode()` dan `Decode()` memiliki beberapa perbedaan. Selain lebar data penampung encode/decode harus dicari terlebih dahulu, terdapat perbedaan lainnya, yaitu pada fungsi ini hasil encode/decode tidak didapat dari nilai kembalian, melainkan dari parameter. Variabel yang digunakan untuk menampung hasil, disisipkan pada parameter fungsi tersebut.

Pada pemanggilan fungsi encode/decode, variabel `encoded` dan `decoded` tidak disisipkan nilai pointer-nya, cukup di-pass dengan cara biasa, tipe datanya sudah dalam bentuk `[]byte`.

### A.43.3. Encode & Decode Data URL

Khusus encode data string yang isinya merupakan URL, lebih efektif menggunakan `URLEncoding` dibandingkan `StdEncoding`.

Cara penerapannya kurang lebih sama, bisa menggunakan metode pertama maupun metode kedua yang sudah dibahas di atas. Cukup ganti `StdEncoding` menjadi `URLEncoding`.

```

var data = "https://kalipare.com/"

var encodedString = base64.URLEncoding.EncodeToString([]byte(data))
fmt.Println(encodedString)

var decodedByte, _ = base64.URLEncoding.DecodeString(encodedString)
var decodedString = string(decodedByte)
fmt.Println(decodedString)

```

## A.44. Hash SHA1

Hash adalah algoritma enkripsi untuk mengubah text menjadi deretan karakter acak. Jumlah karakter hasil hash selalu sama. Hash termasuk *one-way encryption*, membuat hasil dari hash tidak bisa dikembalikan ke text asli.

SHA1 atau **Secure Hash Algorithm 1** merupakan salah satu algoritma hashing yang sering digunakan untuk enkripsi data. Hasil dari sha1 adalah data dengan lebar **20 byte** atau **160 bit**, biasa ditampilkan dalam bentuk bilangan heksadesimal 40 digit.

Di bab ini kita akan belajar tentang pemanfaatan sha1 dan teknik salting dalam hash.

### A.44.1. Penerapan Hash SHA1

Golang menyediakan package `crypto/sha1`, berisikan library untuk keperluan *hashing*. Cara penerapannya cukup mudah, contohnya bisa dilihat pada kode berikut.

```
package main

import "crypto/sha1"
import "fmt"

func main() {
    var text = "this is secret"
    var sha = sha1.New()
    sha.Write([]byte(text))
    var encrypted = sha.Sum(nil)
    var encryptedString = fmt.Sprintf("%x", encrypted)

    fmt.Println(encryptedString)
    // f4ebfd7a42d9a43a536e2bed9ee4974abf8f8dc8
}
```

Variabel hasil dari `sha1.New()` adalah objek bertipe `hash.Hash`, memiliki dua buah method `Write()` dan `Sum()`.

- Method `Write()` digunakan untuk menge-set data yang akan di-hash. Data harus dalam bentuk `[]byte`.
- Method `Sum()` digunakan untuk eksekusi proses hash, menghasilkan data yang sudah di-hash dalam bentuk `[]byte`. Method ini membutuhkan sebuah parameter, isi dengan nil.

Untuk mengambil bentuk heksadesimal string dari data yang sudah di-hash, bisa memanfaatkan fungsi `fmt.Sprintf` dengan layout format `%x`.

```
[novalagung:belajar-golang $ go run bab44.go
original : this is secret
hashed   : f4ebfd7a42d9a43a536e2bed9ee4974abf8f8dc8
novalagung:belajar-golang $ ]
```

### A.44.2. Metode Salting Pada Hash SHA1

Salt dalam konteks kriptografi adalah data acak yang digabungkan pada data asli sebelum proses hash dilakukan.

Hash merupakan enkripsi satu arah dengan lebar data yang sudah pasti, sangat mungkin sekali kalau hasil hash untuk beberapa data adalah sama. Disinilah kegunaan **salt**, teknik ini berguna untuk mencegah serangan menggunakan metode pencocokan data-data yang hasil hash-nya adalah sama (*dictionary attack*).

Langsung saja kita praktikan. Pertama import package yang dibutuhkan. Lalu buat fungsi untuk hash menggunakan salt dari waktu sekarang.

```
package main

import "crypto/sha1"
import "fmt"
import "time"

func doHashUsingSalt(text string) (string, string) {
    var salt = fmt.Sprintf("%d", time.Now().UnixNano())
    var saltedText = fmt.Sprintf("text: '%s', salt: %s", text, salt)
    fmt.Println(saltedText)
    var sha = sha1.New()
    sha.Write([]byte(saltedText))
    var encrypted = sha.Sum(nil)

    return fmt.Sprintf("%x", encrypted), salt
}
```

Salt yang digunakan adalah hasil dari ekspresi `time.Now().UnixNano()`. Hasilnya akan selalu unik setiap detiknya, karena scope terendah waktu pada fungsi tersebut adalah *nano second* atau nano detik.

Selanjutnya test fungsi yang telah dibuat beberapa kali.

```
func main() {
    var text = "this is secret"
    fmt.Printf("original : %s\n\n", text)

    var hashed1, salt1 = doHashUsingSalt(text)
    fmt.Printf("hashed 1 : %s\n\n", hashed1)
    // 929fd8b1e58afca1ebbe30beac3b84e63882ee1a

    var hashed2, salt2 = doHashUsingSalt(text)
    fmt.Printf("hashed 2 : %s\n\n", hashed2)
    // cda603d95286f0aece4b3e1749abe7128a4eed78

    var hashed3, salt3 = doHashUsingSalt(text)
    fmt.Printf("hashed 3 : %s\n\n", hashed3)
    // 9e2b514bc911cb76f7630da50a99d4f4bb200b4

    _, _, _ = salt1, salt2, salt3
}
```

Hasil ekripsi fungsi `doHashUsingSalt` akan selalu beda, karena salt yang digunakan adalah waktu.

```
[novalagung:belajar-golang $ go run bab44.go
original : this is secret

text: 'this is secret', salt: 1444813038167909774
hashed 1 : 3b3b3dc90547617236aeeb8d349eeb79d8b658cb

text: 'this is secret', salt: 1444813038167928750
hashed 2 : f7de5b412793a7f8b11f05849fab18eb137c20ca

text: 'this is secret', salt: 1444813038167934830
hashed 3 : 9ee315e39c142648888054ad1e821e7a21f3a583]
```

Metode ini sering dipakai untuk enkripsi password user. Salt dan data hasil hash harus disimpan pada database, karena digunakan dalam pencocokan password setiap user melakukan login.



## A.45. Arguments & Flag

**Arguments** adalah data opsional yang disisipkan ketika eksekusi program. Sedangkan **flag** merupakan ekstensi dari argument. Dengan flag, penulisan argument menjadi lebih rapi dan terstruktur.

Di bab ini kita akan belajar tentang penggunaan arguments dan flag.

### A.45.1. Penggunaan Arguments

Data arguments bisa didapat lewat variabel `os.Args` (package `os` perlu di-import terlebih dahulu). Data tersebut tersimpan dalam bentuk array dengan pemisah adalah tanda spasi.

Berikut merupakan contoh penggunaannya.

```
package main

import "fmt"
import "os"

func main() {
    var argsRaw = os.Args
    fmt.Printf("-> %#v\n", argsRaw)
    // []string{.../bab45", "banana", "potato", "ice cream"}

    var args = argsRaw[1:]
    fmt.Printf("-> %#v\n", args)
    // []string{"banana", "potato"}
}
```

Pada saat eksekusi program disisipkan juga argument-nya. Sebagai contoh disisipkan 3 buah data sebagai argumen, yaitu: `banana`, `potato`, dan `ice cream`.

Untuk eksekusinya sendiri bisa menggunakan `go run` ataupun dengan cara build-execute.

- Menggunakan `go run`

```
$ go run bab45.go banana potato "ice cream"
```

- Menggunakan `go build`

```
$ go build bab45.go
$ ./bab45 banana potato "ice cream"
```

Variabel `os.Args` mengembalikan tak hanya arguments saja, tapi juga path file executable (jika eksekusi-nya menggunakan `go run` maka path akan merujuk ke folder temporary). Gunakan `os.Args[1:]` untuk mengambil slice arguments-nya saja.

```
[novalagung:belajar-golang $ go run bab45.go banana potato "ice cream"]
-> []string{/var/folders/_2/sdbvcqxd0pq3jz14pwf_rz00000gn/T/go-build918678092
/command-line-arguments/_obj/exe/bab45", "banana", "potato", "ice cream"}
-> []string{"banana", "potato", "ice cream"}
[novalagung:belajar-golang $]
[novalagung:belajar-golang $ go build bab45.go]
[novalagung:belajar-golang $ ./bab45 banana potato "ice cream"]
-> []string{"/bab45", "banana", "potato", "ice cream"}
-> []string{"banana", "potato", "ice cream"}
[novalagung:belajar-golang $]
```

Bisa dilihat pada kode di atas, bahwa untuk data argumen yang ada karakter spasi nya ( ), maka harus diapit tanda petik ( " ), agar tidak dideteksi sebagai 2 argumen.

## A.45.2. Penggunaan Flag

Flag memiliki kegunaan yang sama seperti arguments, yaitu untuk *parameterize* eksekusi program, dengan penulisan dalam bentuk key-value. Berikut merupakan contoh penerapannya.

```
package main

import "flag"
import "fmt"

func main() {
    var name = flag.String("name", "anonymous", "type your name")
    var age = flag.Int64("age", 25, "type your age")

    flag.Parse()
    fmt.Printf("name\t: %s\n", *name)
    fmt.Printf("age\t: %d\n", *age)
}
```

Cara penulisan arguments menggunakan flag:

```
$ go run bab45.go -name="john wick" -age=28
```

Tiap argument harus ditentukan key, tipe data, dan nilai default-nya. Contohnya seperti pada `flag.String()` di atas. Agar lebih mudah dipahami, mari kita bahas kode berikut.

```
var dataName = flag.String("name", "anonymous", "type your name")
fmt.Println(*dataName)
```

Kode tersebut maksudnya adalah, disiapkan flag bertipe `string`, dengan key adalah `name`, dengan nilai default `"anonymous"`, dan keterangan `"type your name"`. Nilai flag nya sendiri akan disimpan kedalam variabel `dataName`.

Nilai balik fungsi `flag.String()` adalah string pointer, jadi perlu di-dereference terlebih dahulu agar bisa mendapatkan nilai aslinya (`*dataName`).

```
[novalagung:belajar-golang $ go run bab45.go -name="john wick" -age=28
name      : john wick
age       : 28
[novalagung:belajar-golang $ go run bab45.go -age=27
name      : anonymous
age       : 27
novalagung:belajar-golang $ ]
```

Flag yang nilainya tidak di set, secara otomatis akan mengembalikan nilai default.

Tabel berikut merupakan macam-macam fungsi flag yang tersedia untuk tiap jenis tipe data.

Nama Fungsi	Return Value
<code>flag.Bool(name, defaultValue, usage)</code>	<code>*bool</code>
<code>flag.Duration(name, defaultValue, usage)</code>	<code>*time.Duration</code>
<code>flag.Float64(name, defaultValue, usage)</code>	<code>*float64</code>
<code>flag.Int(name, defaultValue, usage)</code>	<code>*int</code>
<code>flag.Int64(name, defaultValue, usage)</code>	<code>*int64</code>

<code>flag.String(name, defaultValue, usage)</code>	<code>*string</code>
<code>flag.Uint(name, defaultValue, usage)</code>	<code>*uint</code>
<code>flag.Uint64(name, defaultValue, usage)</code>	<code>*uint64</code>

### A.45.3. Deklarasi Flag Dengan Cara Passing Reference Variabel Penampung Data

Sebenarnya ada 2 cara deklarasi flag yang bisa digunakan, dan cara di atas merupakan cara pertama.

Cara kedua mirip dengan cara pertama, perbedannya adalah kalau di cara pertama nilai pointer flag dikembalikan lalu ditampung variabel; sedangkan pada cara kedua, nilainya diambil lewat parameter pointer.

Agar lebih jelas perhatikan contoh berikut.

```
// cara ke-1
var data1 = flag.String("name", "anonymous", "type your name")
fmt.Println(*data1)

// cara ke-2
var data2 string
flag.StringVar(&data2, "gender", "male", "type your gender")
fmt.Println(data2)
```

Tinggal tambahkan suffix `var` pada pemanggilan nama fungsi flag yang digunakan (contoh `flag.IntVar()`, `flag.BoolVar()`, dll), lalu disisipkan referensi variabel penampung flag sebagai parameter pertama.

Kegunaan dari parameter terakhir method-method flag adalah untuk memunculkan hints atau petunjuk arguments apa saja yang bisa dipakai, ketika argument `--help` ditambahkan saat eksekusi program.

```
[novalagung:chapter-45 $ go build 2-flag.go
[novalagung:chapter-45 $ ./2-flag --help
Usage of ./2-flag:
  -age int
    type your age (default 25)
  -name string
    type your name (default "anonymous")
```

## A.46. Exec

**Exec** digunakan untuk eksekusi perintah command line lewat kode program. Command yang bisa dieksekusi adalah semua command yang bisa dieksekusi di terminal (CMD untuk pengguna Windows).

### A.46.1. Penggunaan Exec

Golang menyediakan package `exec` berisikan banyak fungsi untuk keperluan eksekusi perintah CLI.

Cara untuk eksekusi command cukup mudah, yaitu dengan menuliskan command dalam bentuk string, diikuti arguments-nya (jika ada) sebagai parameter variadic pada fungsi `exec.Command()`.

```
package main

import "fmt"
import "os/exec"

func main() {
    var output1, _ = exec.Command("ls").Output()
    fmt.Printf(" -> ls\n%s\n", string(output1))

    var output2, _ = exec.Command("pwd").Output()
    fmt.Printf(" -> pwd\n%s\n", string(output2))

    var output3, _ = exec.Command("git", "config", "user.name").Output()
    fmt.Printf(" -> git config user.name\n%s\n", string(output3))
}
```

Fungsi `exec.Command()` digunakan untuk menjalankan command. Fungsi tersebut bisa langsung di-chain dengan method `Output()`, jika ingin mendapatkan outputnya. Output yang dihasilkan berbentuk `[]byte`, gunakan cast ke `string` untuk mengambil bentuk string-nya.

```
[novalagung:belajar-golang $ go run bab46.go
 -> ls
bab43.go
bab44.go
bab45.go
bab46.go

-> pwd
/Users/novalagung/Documents/go/src/belajar-golang

-> git config user.name
novalagung]
```

## A.47. File

Ada beberapa cara yang bisa digunakan untuk operasi file di Golang. Pada bab ini kita akan mempelajari teknik yang paling dasar, yaitu dengan memanfaatkan `os.File`.

### A.47.1. Membuat File Baru

Pembuatan file di Golang sangatlah mudah, cukup dengan memanggil fungsi `os.Create()` lalu memasukkan path file ingin dibuat sebagai parameter fungsi tersebut.

Jika file yang akan dibuat sudah ada, maka akan ditimpa. Bisa memanfaatkan `os.IsNotExist()` untuk mendeteksi apakah file sudah dibuat atau belum.

Berikut merupakan contoh pembuatan file.

```
package main

import "fmt"
import "os"

var path = "/Users/novalagung/Documents/temp/test.txt"

func isError(err error) bool {
    if err != nil {
        fmt.Println(err.Error())
    }

    return (err != nil)
}

func createFile() {
    // deteksi apakah file sudah ada
    var _, err = os.Stat(path)

    // buat file baru jika belum ada
    if os.IsNotExist(err) {
        var file, err = os.Create(path)
        if isError(err) { return }
        defer file.Close()
    }

    fmt.Println("==> file berhasil dibuat", path)
}

func main() {
    createFile()
}
```

Fungsi `os.Stat()` mengembalikan 2 data, yaitu informasi tentang path yang dicari, dan error (jika ada). Masukkan error kembalian fungsi tersebut sebagai parameter fungsi `os.IsNotExist()`, untuk mendeteksi apakah file yang akan dibuat sudah ada. Jika belum ada, maka fungsi tersebut akan mengembalikan nilai `true`.

Fungsi `os.Create()` digunakan untuk membuat file pada path tertentu. Fungsi ini mengembalikan objek `*os.File` dari file yang bersangkutan. File yang baru terbuat statusnya adalah otomatis **open**, maka dari itu perlu untuk di-**close** menggunakan method `file.Close()` setelah file tidak digunakan lagi.

Membuka file terbuka ketika sudah tak lagi digunakan bukan hal yang baik, karena efeknya ke memory dan akses ke file itu sendiri, file akan di-lock sehingga tidak bisa digunakan oleh proses lain selama status file masih open atau belum di-close.

```
[novalagung:chapter-47 $ ls /Users/novalagung/Documents/temp/
[novalagung:chapter-47 $ go run 1-membuat-file.go
==> file berhasil dibuat /Users/novalagung/Documents/temp/test.txt
[novalagung:chapter-47 $ ls /Users/novalagung/Documents/temp/
test.txt
```

## A.47.2. Mengedit Isi File

Untuk mengedit file, yang perlu dilakukan pertama adalah membuka file dengan level akses **write**. Setelah mendapatkan objek file-nya, gunakan method `WriteString()` untuk pengisian data. Terakhir panggil method `Sync()` untuk menyimpan perubahan.

```
func writeFile() {
    // buka file dengan level akses READ & WRITE
    var file, err = os.OpenFile(path, os.O_RDWR, 0644)
    if isError(err) { return }
    defer file.Close()

    // tulis data ke file
    _, err = file.WriteString("halo\n")
    if isError(err) { return }
    _, err = file.WriteString("mari belajar golang\n")
    if isError(err) { return }

    // simpan perubahan
    err = file.Sync()
    if isError(err) { return }

    fmt.Println("==> file berhasil di isi")
}

func main() {
    writeFile()
}
```

Pada program di atas, file dibuka dengan level akses **read** dan **write** dengan kode permission **0664**. Setelah itu, beberapa string diisikan kedalam file tersebut menggunakan `WriteString()`. Di akhir, semua perubahan terhadap file akan disimpan dengan dipanggilnya `Sync()`.

```
[novalagung:chapter-47 $ cat /Users/novalagung/Documents/temp/test.txt
[novalagung:chapter-47 $ go run 2-mengedit-file.go
==> file berhasil di isi
[novalagung:chapter-47 $ cat /Users/novalagung/Documents/temp/test.txt
halo
mari belajar golang
novalagung:chapter-47 $ ]
```

## A.47.3. Membaca Isi File

File yang ingin dibaca harus dibuka terlebih dahulu menggunakan fungsi `os.OpenFile()` dengan level akses minimal adalah **read**. Setelah itu, gunakan method `Read()` dengan parameter adalah variabel, yang dimana hasil proses baca akan disimpan ke variabel tersebut.

```
// tambahkan di bagian import package io
import "io"
```

```

func readFile() {
    // buka file
    var file, err = os.OpenFile(path, os.O_RDWR, 0644)
    if isError(err) { return }
    defer file.Close()

    // baca file
    var text = make([]byte, 1024)
    for {
        n, err := file.Read(text)
        if err != io.EOF {
            if isError(err) { break }
        }
        if n == 0 {
            break
        }
    }
    if isError(err) { return }

    fmt.Println("==> file berhasil dibaca")
    fmt.Println(string(text))
}

func main() {
    readFile()
}

```

Pada kode di atas `os.OpenFile()` digunakan untuk membuka file. Fungsi tersebut memiliki beberapa parameter.

1. Parameter pertama adalah path file yang akan dibuka.
2. Parameter kedua adalah level akses. `os.O_RDONLY` maksudnya adalah **read only**.
3. Parameter ketiga adalah permission file-nya.

Variabel `text` disiapkan bertipe slice `[]byte` dengan alokasi elemen 1024. Variabel tersebut bertugas menampung data hasil statement `file.Read()`. Proses pembacaan file akan dilakukan terus menerus, berurutan dari baris pertama hingga akhir.

Error yang muncul ketika eksekusi `file.Read()` akan di-filter, ketika error tersebut adalah selain `io.EOF` maka proses baca file akan berlanjut. Error `io.EOF` sendiri menandakan bahwa file yang sedang dibaca adalah baris terakhir isi atau **end of file**.

```
[novalagung:chapter-47 $ go run 3-membaca-file.go
==> file berhasil dibaca
halo
mari belajar golang
```

## A.47.4. Menghapus File

Cara menghapus file sangatlah mudah, cukup panggil fungsi `os.Remove()`, masukan path file yang ingin dihapus sebagai parameter.

```

func deleteFile() {
    var err = os.Remove(path)
    if isError(err) { return }

    fmt.Println("==> file berhasil di delete")
}

func main() {
    deleteFile()
}

```

```
[novalagung:chapter-47 $ ls /Users/novalagung/Documents/temp/
test.txt
[novalagung:chapter-47 $ go run 4-menghapus-file.go
==> file berhasil di delete
[novalagung:chapter-47 $ ls /Users/novalagung/Documents/temp/
novalagung:chapter-47 $ ]
```

## A.48. Web

Golang menyediakan package `net/http`, berisi berbagai macam fitur untuk keperluan pembuatan aplikasi berbasis web. Termasuk didalamnya routing, server, templating, dan lainnya, semua tinggal pakai.

Golang memiliki web server sendiri, dan web server tersebut berada di dalam golang, tidak seperti bahasa lain yang server nya terpisah dan perlu di-instal sendiri (seperti PHP yang memerlukan Apache, .NET yang memerlukan IIS).

Di bab ini kita akan belajar cara pembuatan aplikasi web sederhana dan pemanfaatan template untuk mendesain view.

### A.48.1. Membuat Aplikasi Web Sederhana

Package `net/http` memiliki banyak sekali fungsi yang bisa dimanfaatkan. Di bagian ini kita akan mempelajari beberapa fungsi penting seperti *routing* dan *start server*.

Program dibawah ini merupakan contoh sederhana untuk memunculkan text di web ketika url tertentu diakses.

```
package main

import "fmt"
import "net/http"

func index(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintln(w, "apa kabar!")
}

func main() {
    http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
        fmt.Fprintln(w, "halo!")
    })

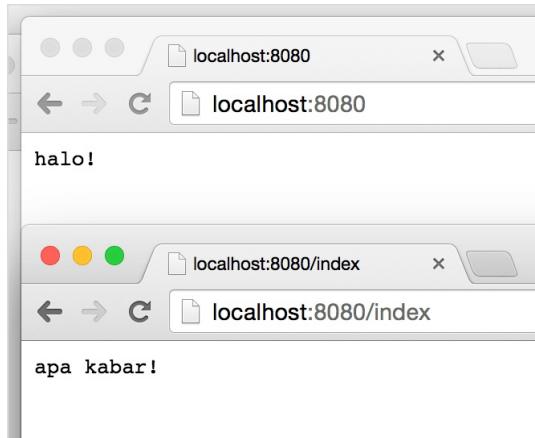
    http.HandleFunc("/index", index)

    fmt.Println("starting web server at http://localhost:8080/")
    http.ListenAndServe(":8080", nil)
}
```

Jalankan program tersebut.

```
[novalagung:belajar-golang $ go run bab48.go
starting web server at http://localhost:8080/
]
```

Jika muncul dialog **Do you want the application “bab48” to accept incoming network connections?** atau sejenis, pilih allow. Setelah itu, buka url <http://localhost/> dan <http://localhost/index> lewat browser.



Fungsi `http.HandleFunc()` digunakan untuk routing aplikasi web. Maksud dari routing adalah penentuan aksi ketika url tertentu diakses oleh user.

Pada kode di atas 2 rute didaftarkan, yaitu `/` dan `/index`. Aksi dari rute `/` adalah menampilkan text `"halo"` di halaman website. Sedangkan `/index` menampilkan text `"apa kabar!"`.

Fungsi `http.HandleFunc()` memiliki 2 buah parameter yang harus diisi. Parameter pertama adalah rute yang diinginkan. Parameter kedua adalah *callback* atau aksi ketika rute tersebut diakses. Callback tersebut bertipe fungsi `func(w http.ResponseWriter, r *http.Request)`.

Pada pendaftaran rute `/index`, callback-nya adalah fungsi `index()`, hal seperti ini diperbolehkan asalkan tipe dari fungsi tersebut sesuai.

Fungsi `http.ListenAndServe()` digunakan untuk menghidupkan server sekaligus menjalankan aplikasi menggunakan server tersebut. Di golang, 1 web aplikasi adalah 1 buah server berbeda.

Pada contoh di atas, server dijalankan pada port `8080`.

Perlu diingat, setiap ada perubahan pada file `.go`, `go run` harus dipanggil lagi.

Untuk menghentikan web server, tekan **CTRL+C** pada terminal atau CMD, dimana pengeksekusian aplikasi berlangsung.

## A.48.2. Penggunaan Template Web

Template egninmemberikan kemudahan dalam mendesain tampilan view aplikasi website. Dan kabar baiknya golang menyediakan engine template sendiri, dengan banyak fitur yang tersedia didalamnya.

Di sini kita akan belajar contoh sederhana penggunaan template untuk menampilkan data. Pertama siapkan dahulu template nya. Buat file `template.html` lalu isi dengan kode berikut.

```
<html>
  <head>
    <title>Golang learn net/http</title>
  </head>
  <body>
    <p>Hello {{.Name}} !</p>
    <p>{{.Message}}</p>
  </body>
</html>
```

Kode `{{.Name}}` artinya memunculkan isi data property `Name` yang dikirim dari router. Kode tersebut nantinya di-replace dengan isi variabel `Name`.

Selanjutnya ubah isi file `.go` dengan kode berikut.

```
package main

import "fmt"
import "html/template"
import "net/http"

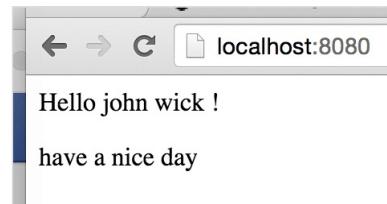
func main() {
    http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
        var data = map[string]string{
            "Name":      "john wick",
            "Message":   "have a nice day",
        }

        var t, err = template.ParseFiles("template.html")
        if err != nil {
            fmt.Println(err.Error())
            return
        }

        t.Execute(w, data)
    })

    fmt.Println("starting web server at http://localhost:8080/")
    http.ListenAndServe(":8080", nil)
}
```

Jalankan, lalu buka <http://localhost:8080/>, maka data `name` dan `message` akan muncul di view.



Fungsi `template.ParseFiles()` digunakan untuk parsing template, mengembalikan 2 data yaitu instance template-nya dan error (jika ada). Pemanggilan method `Execute()` akan membuat hasil parsing template ditampilkan ke layar web browser.

Pada kode di atas, variabel `data` disisipkan sebagai parameter ke-2 method `Execute()`. Isi dari variabel tersebut bisa diakses di-view dengan menggunakan notasi `{{.NAME_PROPERTY}}` (nama variabel sendiri tidak perlu dituliskan, langsung nama property didalamnya).

Pada contoh di atas, statement di view `{{.Name}}` akan menampilkan isi dari `data.Name`.

## A.49. URL Parsing

Data string url bisa dikonversi kedalam bentuk `url.URL`. Dengan menggunakan tipe tersebut akan ada banyak informasi yang bisa kita manfaatkan, diantaranya adalah jenis protokol yang digunakan, path yang diakses, query, dan lainnya.

Berikut adalah contoh sederhana konversi string ke `url.URL`.

```
package main

import "fmt"
import "net/url"

func main() {
    var urlString = "http://depeloper.com:80/hello?name=john wick&age=27"
    var u, e = url.Parse(urlString)
    if e != nil {
        fmt.Println(e.Error())
        return
    }

    fmt.Printf("url: %s\n", urlString)

    fmt.Printf("protocol: %s\n", u.Scheme) // http
    fmt.Printf("host: %s\n", u.Host)       // depeloper.com:80
    fmt.Printf("path: %s\n", u.Path)       // /hello

    var name = u.Query()["name"][0] // john wick
    var age = u.Query()["age"][0]   // 27
    fmt.Printf("name: %s, age: %s\n", name, age)
}
```

Fungsi `url.Parse()` digunakan untuk parsing string ke bentuk url. Mengembalikan 2 data, variabel objek bertipe `url.URL` dan error (jika ada). Lewat variabel objek tersebut pengaksesan informasi url akan menjadi lebih mudah, contohnya seperti nama host bisa didapatkan lewat `u.Host`, protokol lewat `u.Scheme`, dan lainnya.

Selain itu, query yang ada pada url akan otomatis diparsing juga, menjadi bentuk `map[string][]string`, dengan key adalah nama elemen query, dan value array string yang berisikan value elemen query.

```
[novalagung:belajar-golang $ go run bab49.go
url: http://localhost:8080/hello?name=john wick&age=27
protocol: http
host: localhost:8080
path: /hello
name: john wick, age: 27
novalagung:belajar-golang $ ]
```

## A.50. JSON

**JSON** atau *Javascript Object Notation* adalah notasi standar yang umum digunakan untuk komunikasi data via web. JSON merupakan subset dari *javascript*.

Golang menyediakan package `encoding/json` yang berisikan banyak fungsi untuk kebutuhan operasi json.

Di bab ini, kita akan belajar cara untuk konverstion string yang berbentuk json menjadi objek golang, dan sebaliknya.

### A.50.1. Decode JSON Ke Variabel Objek Cetakan Struct

Di Golang data json tipenya adalah `string`. Dengan menggunakan `json.Unmarshal`, json string bisa dikonversi menjadi bentuk objek, entah itu dalam bentuk `map[string]interface{}` ataupun variabel objek hasil `struct`.

Program berikut ini adalah contoh cara decoding json ke bentuk objek. Pertama import package yang dibutuhkan, lalu siapkan struct `User`.

```
package main

import "encoding/json"
import "fmt"

type User struct {
    FullName string `json:"Name"`
    Age      int
}
```

Hasil decode nantinya akan disimpan ke variabel objek cetakan struct `User`.

Selanjutnya siapkan data json string sederhan. Perlu dilakukan casting ke tipe `[]byte`, karena fungsi `json.Unmarshal` hanya menerima data bertipe `[]byte`.

```
func main() {
    var jsonString = `{"Name": "john wick", "Age": 27}`
    var jsonData = []byte(jsonString)

    var data User

    var err = json.Unmarshal(jsonData, &data)
    if err != nil {
        fmt.Println(err.Error())
        return
    }

    fmt.Println("user :", data.FullName)
    fmt.Println("age  :", data.Age)
}
```

Dalam penggunaan fungsi `json.Unmarshal`, variabel penampung hasil decode harus di-pass dalam bentuk pointer, contohnya seperti `&data`.

```
[novalagung:belajar-golang $ go run bab50.go
user : john wick
age  : 27
novalagung:belajar-golang $ ]
```

Bisa dilihat bahwa salah satu property struct `User`, yaitu `FullName` memiliki **tag** `json:"Name"`. Tag tersebut digunakan untuk mapping data json ke property yang bersangkutan.

Data json yang akan diparsing memiliki 2 property yaitu `Name` dan `Age`. Kebetulan penulisan `Age` pada data json dan pada struktur struct adalah sama, berbeda dengan `Name` yang tidak ada pada struct.

Property `FullName` struct tersebut kemudian ditugaskan untuk menampung data json property `Name`, ditandai dengan penambahan tag `json:"Name"` pada saat deklarasi struct-nya.

Perlu diketahui bahwa untuk decode data json ke variabel objek hasil struct, semua level akses property struct-nya harus publik.

## A.50.2. Decode JSON Ke `map[string]interface{}` & interface{}``

Tak hanya ke objek cetakan struct, target decoding data json juga bisa berupa variabel bertipe `map[string]interface{}``.

```
var data1 map[string]interface{}
json.Unmarshal(jsonData, &data1)

fmt.Println("user :", data1["Name"])
fmt.Println("age :", data1["Age"])
```

Variabel bertipe `interface{}`` juga bisa digunakan untuk menampung hasil decode. Dengan catatan pada pengaksesan nilai property, harus dilakukan casting terlebih dahulu ke `map[string]interface{}``.

```
var data2 interface{}
json.Unmarshal(jsonData, &data2)

var decodedData = data2.(map[string]interface{})
fmt.Println("user :", decodedData["Name"])
fmt.Println("age :", decodedData["Age"])
```

## A.50.3. Decode Array JSON Ke Array Objek

Decode data dari array json ke slice/array objek masih sama, siapkan saja variabel penampung hasil decode dengan tipe slice struct. Contohnya bisa dilihat pada kode berikut.

```
var jsonString = `[
    {"Name": "john wick", "Age": 27},
    {"Name": "ethan hunt", "Age": 32}
]` 

var data []User

var err = json.Unmarshal([]byte(jsonString), &data)
if err != nil {
    fmt.Println(err.Error())
    return
}

fmt.Println("user 1:", data[0].FullName)
fmt.Println("user 2:", data[1].FullName)
```

## A.50.4. Encode Objek Ke JSON

Setelah sebelumnya dijelaskan beberapa cara decode data dari json ke objek, sekarang kita akan belajar cara **encode** data objek ke bentuk json string.

Fungsi `json.Marshal` digunakan untuk decoding data ke json string. Sumber data bisa berupa variabel objek cetakan struct, `map[string]interface{}`, atau slice.

Pada contoh berikut, data slice struct dikonversi ke dalam bentuk json string. Hasil konversi berupa `[]byte`, casting terlebih dahulu ke tipe `string` agar bisa ditampilkan bentuk json-nya.

```
var object = []User{{"john wick", 27}, {"ethan hunt", 32}}
var jsonData, err = json.Marshal(object)
if err != nil {
    fmt.Println(err.Error())
    return
}

var jsonString = string(jsonData)
fmt.Println(jsonString)
```

Output:

```
[novalagung:belajar-golang $ go run bab50.go
[{"Name":"john wick","Age":27}, {"Name":"ethan hunt","Age":32}]
novalagung:belajar-golang $ ]
```

## A.51. Web API JSON

Pada bab ini kita akan mengkombinasikan pembahasan 2 bab sebelumnya, yaitu web dan JSON, untuk membuat sebuah web API dengan tipe data reponse berbentuk JSON.

Web API adalah sebuah web yang menerima request dari client dan menghasilkan response, biasa berupa JSON/XML. Di bab ini kita akan buat tanpa authentikasi.

### A.51.1. Pembuatan Web API

Pertama siapkan terlebih dahulu struct dan beberapa data sample.

```
package main

import "encoding/json"
import "net/http"
import "fmt"

type student struct {
    ID      string
    Name    string
    Grade   int
}

var data = []student{
    student{"E001", "ethan", 21},
    student{"W001", "wick", 22},
    student{"B001", "bourne", 23},
    student{"B002", "bond", 23},
}
```

Struct `student` di atas digunakan sebagai tipe elemen array sample data, ditampung variabel `data`.

Selanjutnya buat fungsi `users()` untuk handle route `/users`. Didalam fungsi tersebut ada proses deteksi jenis request lewat property `r.Method()`, untuk mencari tahu apakah jenis request adalah **POST** atau **GET** atau lainnya.

```
func users(w http.ResponseWriter, r *http.Request) {
    w.Header().Set("Content-Type", "application/json")

    if r.Method == "POST" {
        var result, err = json.Marshal(data)

        if err != nil {
            http.Error(w, err.Error(), http.StatusInternalServerError)
            return
        }

        w.Write(result)
        return
    }

    http.Error(w, "", http.StatusBadRequest)
}
```

Jika request adalah POST, maka data yang di-encode ke JSON dijadikan sebagai response.

Statement `w.Header().Set("Content-Type", "application/json")` digunakan untuk menentukan tipe response, yaitu sebagai JSON. Sedangkan `r.Write()` digunakan untuk mendaftarkan data sebagai response.

Selebihnya, jika request tidak valid, response di set sebagai error menggunakan fungsi `http.Error()`.

Siapkan juga handler untuk rute `/user`. Perbedaan rute ini dengan rute `/users` di atas adalah:

- `/users` menghasilkan semua sample data yang ada (array).
- `/user` menghasilkan satu buah data saja, diambil dari data sample berdasarkan `ID`-nya. Pada rute ini, client harus mengirimkan juga informasi `ID` data yang dicari.

```
func user(w http.ResponseWriter, r *http.Request) {
    w.Header().Set("Content-Type", "application/json")

    if r.Method == "POST" {
        var id = r.FormValue("id")
        var result []byte
        var err error

        for _, each := range data {
            if each.ID == id {
                result, err = json.Marshal(each)

                if err != nil {
                    http.Error(w, err.Error(), http.StatusInternalServerError)
                    return
                }

                w.Write(result)
                return
            }
        }

        http.Error(w, "User not found", http.StatusBadRequest)
        return
    }

    http.Error(w, "", http.StatusBadRequest)
}
```

Method `r.FormValue()` digunakan untuk mengambil data form yang dikirim dari client, pada konteks ini data yang dimaksud adalah `ID`.

Dengan menggunakan `ID` tersebut dicarilah data yang relevan. Jika ada, maka dikembalikan sebagai response. Jika tidak ada maka error **400, Bad Request** dikembalikan dengan pesan **User Not Found**.

Terakhir, implementasikan kedua handler di atas.

```
func main() {
    http.HandleFunc("/users", users)
    http.HandleFunc("/user", user)

    fmt.Println("starting web server at http://localhost:8080/")
    http.ListenAndServe(":8080", nil)
}
```

Jalankan program, sekarang web server sudah live dan bisa dikonsumsi datanya.

```
[novalagung:belajar-golang $ go run bab51.go
starting web server at http://localhost:8080/
]
```

## A.51.2. Test API

Setelah web server sudah berjalan, web API yang telah dibuat perlu untuk di tes. Di sini saya menggunakan Google Chrome plugin bernama [Postman](#) untuk mengetes API yang sudah dibuat.

- Test `/users`, apakah data yang dikembalikan sudah benar.

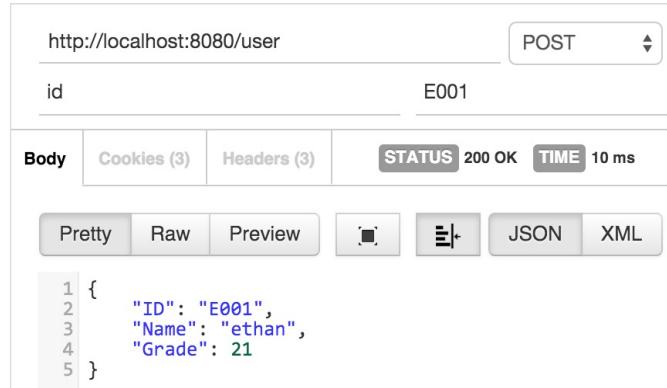


```

1 [ 
2   {
3     "ID": "E001",
4     "Name": "ethan",
5     "Grade": 21
6   },
7   {
8     "ID": "W001",
9     "Name": "wick",
10    "Grade": 22
11  },
12  {
13    "ID": "B001",
14    "Name": "bourne",
15    "Grade": 23
16  },
17  {
18    "ID": "B002",
19    "Name": "bond",
20    "Grade": 23
21  }
22 ]

```

- Test `/user`, isi form data `id` dengan nilai `E001`.



```

1 {
2   "ID": "E001",
3   "Name": "ethan",
4   "Grade": 21
5 }

```

## A.52. HTTP Request

Di bab sebelumnya kita telah belajar tentang bagaimana membuat Web API yang mem-provide data JSON, pada bab ini kita akan belajar mengenai cara untuk mengkonsumsi data tersebut.

Pastikan anda sudah mempraktekkan apa-apa yang ada pada bab sebelumnya (bab 51), karena web api server yang sudah dibuat pada bab sebelumnya kita juga pada bab ini.

```
[novalagung:belajar-golang $ go run bab51.go
starting web server at http://localhost:8080/]
```

### A.52.1. Penggunaan HTTP Request

Package `net/http`, selain berisikan tools untuk keperluan pembuatan web, juga berisikan fungsi-fungsi untuk melakukan http request. Salah satunya adalah `http.NewRequest()` yang akan kita bahas di sini.

Sebelumnya, import package yang dibutuhkan. Dan siapkan struct `student` yang nantinya akan dipakai sebagai tipe data reponse dari web API. Struk tersebut skema nya sama dengan yang ada pada bab 51.

```
package main

import "fmt"
import "net/http"
import "encoding/json"

var baseURL = "http://localhost:8080"

type student struct {
    ID   string
    Name string
    Grade int
}
```

Setelah itu buat fungsi `fetchUsers()`. Fungsi ini bertugas melakukan request ke <http://localhost:8080/users>, menerima response dari request tersebut, lalu menampilkannya.

```
func fetchUsers() ([]student, error) {
    var err error
    var client = &http.Client{}
    var data []student

    request, err := http.NewRequest("POST", baseURL+"/users", nil)
    if err != nil {
        return nil, err
    }

    response, err := client.Do(request)
    if err != nil {
        return nil, err
    }
    defer response.Body.Close()

    err = json.NewDecoder(response.Body).Decode(&data)
    if err != nil {
        return nil, err
    }

    return data, nil
}
```

```
}
```

Statement `&http.Client{}` menghasilkan instance `http.Client`. Objek ini nantinya diperlukan untuk eksekusi request.

Fungsi `http.NewRequest()` digunakan untuk membuat request baru. Fungsi tersebut memiliki 3 parameter yang wajib diisi.

1. Parameter pertama, berisikan tipe request **POST** atau **GET** atau lainnya
2. Parameter kedua, adalah URL tujuan request
3. Parameter ketiga, form data request (jika ada)

Fungsi tersebut menghasilkan instance bertipe `http.Request`. Objek tersebut nantinya disisipkan pada saat eksekusi request.

Cara eksekusi request sendiri adalah dengan memanggil method `Do()` pada instance `http.Client` yang sudah dibuat, dengan parameter adalah instance request-nya. Contohnya seperti pada `client.Do(request)`.

Method tersebut mengembalikan instance bertipe `http.Response`, yang didalamnya berisikan informasi yang dikembalikan dari web API.

Data response bisa diambil lewat property `Body` dalam bentuk string. Gunakan JSON Decoder untuk mengkonversinya menjadi bentuk JSON. Contohnya bisa dilihat di kode di atas,  
`json.NewDecoder(response.Body).Decode(&data)`. Setelah itu barulah kita bisa menampilkannya.

Perlu diketahui, data response perlu di-**close** setelah tidak dipakai. Caranya seperti pada kode `defer response.Body.Close()`.

Implementasikan fungsi `fetchUsers()` di atas pada `main`.

```
func main() {
    var users, err = fetchUsers()
    if err != nil {
        fmt.Println("Error!", err.Error())
        return
    }

    for _, each := range users {
        fmt.Printf("ID: %s\t Name: %s\t Grade: %d\n", each.ID, each.Name, each.Grade)
    }
}
```

Jalankan program untuk mengetes hasilnya.

```
[novalagung:belajar-golang $ go run bab52.go
ID: E001      Name: ethan      Grade: 21
ID: W001      Name: wick      Grade: 22
ID: B001      Name: bourne    Grade: 23
ID: B002      Name: bond      Grade: 23
novalagung:belajar-golang $ ]
```

### A.52.3. HTTP Request Dengan Form Data

Untuk menyisipkan data pada sebuah request, ada beberapa hal yang perlu ditambahkan. Yang pertama, import beberapa package lagi, `bytes` dan `net/url`.

```
import "bytes"
import "net/url"
```

Buat fungsi baru, isinya request ke <http://localhost:8080/user> dengan data yang disisipkan adalah `ID`.

```
func fetchUser(ID string) (student, error) {
    var err error
    var client = &http.Client{}
    var data student

    var param = url.Values{}
    param.Set("id", ID)
    var payload = bytes.NewBufferString(param.Encode())

    request, err := http.NewRequest("POST", baseURL+"/user", payload)
    if err != nil {
        return data, err
    }
    request.Header.Set("Content-Type", "application/x-www-form-urlencoded")

    response, err := client.Do(request)
    if err != nil {
        return data, err
    }
    defer response.Body.Close()

    err = json.NewDecoder(response.Body).Decode(&data)
    if err != nil {
        return data, err
    }

    return data, nil
}
```

Isi fungsi di atas bisa dilihat memiliki beberapa kemiripan dengan fungsi `fetchusers()` sebelumnya.

Statement `url.Values{}` akan menghasilkan objek yang nantinya digunakan sebagai form data request. Pada objek tersebut perlu di set data apa saja yang ingin dikirimkan menggunakan fungsi `Set()` seperti pada `param.Set("id", ID)`.

Statement `bytes.NewBufferString(param.Encode())` maksudnya, objek form data di-encode lalu diubah menjadi bentuk `bytes.Buffer`, yang nantinya disisipkan pada parameter ketiga pemanggilan fungsi `http.NewRequest()`.

Karena data yang akan dikirim di-encode, maka pada header perlu di set tipe konten request-nya. Kode `request.Header.Set("Content-Type", "application/x-www-form-urlencoded")` artinya tipe konten request di set sebagai `application/x-www-form-urlencoded`.

Pada konteks HTML, HTTP Request yang di trigger dari tag `<form></form>` secara default tipe kontennya sudah di set `application/x-www-form-urlencoded`. Lebih detailnya bisa merujuk ke spesifikasi HTML form <http://www.w3.org/TR/html401/interact/forms.html#h-17.13.4.1>

Response dari rute `/user` bukan berupa array objek, melainkan sebuah objek. Maka pada saat decode pastikan tipe variabel penampung hasil decode data response adalah `student` (bukan `[]student`).

Terakhir buat implementasinya pada fungsi `main`.

```
func main() {
    var user1, err = fetchUser("E001")
    if err != nil {
        fmt.Println("Error!", err.Error())
        return
    }

    fmt.Printf("ID: %s\t Name: %s\t Grade: %d\n", user1.ID, user1.Name, user1.Grade)
}
```

Pada kode di atas `ID` ditentukan nilainya `"E001"`. Jalankan program untuk mengetes apakah data yang dikembalikan sesuai.

```
[novalagung:belajar-golang $ go run bab52.go
ID: E001      Name: ethan      Grade: 21
novalagung:belajar-golang $ ]
```

## A.53. SQL

Golang menyediakan package `database/sql` berisikan generic interface untuk keperluan interaksi dengan database sql. Package ini hanya bisa digunakan ketika **driver** database engine yang dipilih juga ada.

Ada cukup banyak sql driver yang tersedia untuk Golang, detailnya bisa diakses di <https://github.com/golang/go/wiki/SQLDrivers>. Beberapa diantaranya:

- MySql
- Oracle
- MS Sql Server
- dan lainnya

Driver-driver tersebut merupakan project open source yang diinisiasi oleh komunitas di Github. Artinya kita selaku developer juga bisa ikut berkontribusi didalamnya.

Pada bab ini kita akan belajar bagaimana berkomunikasi dengan database MySQL menggunakan driver [Go MySQL Driver](#).

### A.53.1. Instalasi Driver

Unduh driver mysql menggunakan `go get`.

```
go get github.com/go-sql-driver/mysql
```

```
[novalagung:belajar-golang $ go get github.com/go-sql-driver/mysql
[novalagung:belajar-golang $ ls $GOPATH/src/github.com/go-sql-driver/mysql
 AUTHORS          collations.go      packets.go
 CHANGELOG.md    connection.go     result.go
 CONTRIBUTING.md const.go        rows.go
 LICENSE          driver.go       statement.go
 README.md        driver_test.go   transaction.go
 appengine.go     errors.go      utils.go
 benchmark_test.go errors_test.go utils_test.go
 buffer.go        infile.go
 novalagung:belajar-golang $ ]
```

### A.53.2. Setup Database

Sebelumnya pastikan sudah ada [mysql server](#) yang terinstal di lokal anda.

Buat database baru bernama `db_belajar_golang`, dan tabel baru bernama `tb_student`.

```
CREATE TABLE IF NOT EXISTS `tb_student` (
  `id` varchar(5) NOT NULL,
  `name` varchar(255) NOT NULL,
  `age` int(11) NOT NULL,
  `grade` int(11) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

INSERT INTO `tb_student`(`id`, `name`, `age`, `grade`) VALUES
('B001', 'Jason Bourne', 29, 1),
('B002', 'James Bond', 27, 1),
('E001', 'Ethan Hunt', 27, 2),
('W001', 'John Wick', 28, 2);

ALTER TABLE `tb_student` ADD PRIMARY KEY (`id`);
```

### A.53.3. Membaca Data Dari MySQL Server

Import package yang dibutuhkan, lalu disiapkan struct dengan skema yang sama seperti pada tabel `tb_student` di database. Nantinya struct ini digunakan sebagai tipe data penampung hasil query.

```
package main

import "fmt"
import "database/sql"
import _ "github.com/go-sql-driver/mysql"

type student struct {
    id   string
    name string
    age  int
    grade int
}
```

Driver database yang digunakan perlu di-import menggunakan tanda `_`, karena meskipun dibutuhkan oleh package `database/sql`, kita tidak langsung berinteraksi dengan driver tersebut.

Selanjutnya buat fungsi untuk koneksi ke database.

```
func connect() (*sql.DB, error) {
    db, err := sql.Open("mysql", "root:@tcp(127.0.0.1:3306)/db_belajar_golang")
    if err != nil {
        return nil, err
    }

    return db, nil
}
```

Fungsi `sql.Open()` digunakan untuk memulai koneksi dengan database. Fungsi tersebut memiliki 2 parameter mandatory, nama driver dan **connection string**.

Skema connection string untuk driver mysql yang kita gunakan cukup unik, `root@tcp(127.0.0.1:3306)/db_belajar_golang`. Dibawah ini merupakan skema connection string yang bisa digunakan pada driver Go MySQL Driver. Jika anda menggunakan driver mysql lain, skema koneksinya bisa saja berbeda tergantung driver yang digunakan.

```
user:password@tcp(host:port)/dbname
user@tcp(host:port)/dbname
```

Di bawah ini adalah penjelasan mengenai connection string yang digunakan pada fungsi `connect()`.

```
root@tcp(127.0.0.1:3306)/db_belajar_golang
// user      => root
// password =>
// host      => 127.0.0.1 atau localhost
// port      => 3306
// dbname    => db_belajar_golang
```

Setelah fungsi untuk koneksi dengan database sudah dibuat, saatnya untuk mempraktekan proses pembacaan data dari server database. Siapkan fungsi `sqlQuery()` dengan isi adalah kode berikut.

```
func sqlQuery() {
```

```

db, err := connect()
if err != nil {
    fmt.Println(err.Error())
    return
}
defer db.Close()

var age = 27
rows, err := db.Query("select id, name, grade from tb_student where age = ?", age)
if err != nil {
    fmt.Println(err.Error())
    return
}
defer rows.Close()

var result []student

for rows.Next() {
    var each = student{}
    var err = rows.Scan(&each.id, &each.name, &each.grade)

    if err != nil {
        fmt.Println(err.Error())
        return
    }

    result = append(result, each)
}

if err = rows.Err(); err != nil {
    fmt.Println(err.Error())
    return
}

for _, each := range result {
    fmt.Println(each.name)
}
}

```

Setiap kali terbuat koneksi baru, jangan lupa untuk selalu **close** instance koneksinya. Bisa menggunakan keyword `defer` seperti pada kode di atas, `defer db.Close()`.

Fungsi `db.Query()` digunakan untuk eksekusi sql query. Fungsi tersebut parameter keduanya adalah variadic, sehingga boleh tidak diisi. Pada kode di atas bisa dilihat bahwa nilai salah satu clause `where` adalah tanda tanya (`?`). Tanda tersebut kemudian akan ter-replace oleh nilai pada parameter setelahnya (nilai variabel `age`). Teknik penulisan query sejenis ini sangat dianjurkan, untuk mencegah [sql injection](#).

Fungsi tersebut menghasilkan instance bertipe `sql.*Rows`, yang juga perlu di **close** ketika sudah tidak digunakan (`defer rows.Close()`).

Selanjutnya, sebuah array dengan tipe elemen struct `student` disiapkan dengan nama `result`. Nantinya hasil query akan ditampung ke variabel tersebut.

Kemudian dilakukan perulangan dengan acuan kondisi adalah `rows.Next()`. Perulangan dengan cara ini dilakukan sebanyak jumlah total record yang ada, berurutan dari record pertama hingga akhir, satu per satu.

Method `Scan()` milik `sql.Rows` berfungsi untuk mengambil nilai record yang sedang diiterasi, untuk disimpan pada variabel pointer. Variabel yang digunakan untuk menyimpan field-field record dituliskan berurutan sebagai parameter variadic, sesuai dengan field yang di select pada query. Silakan lihat perbandingan dibawah ini untuk lebih jelasnya.

```

// query
select id, name, grade ...

// scan

```

```
rows.Scan(&each.id, &each.name, &each.grade ...
```

Data record yang didapat kemudian di-append ke slice `result`, lewat statement `result = append(result, each)`.

OK, sekarang tinggal panggil fungsi `sqlQuery()` di `main`, lalu jalankan program.

```
func main() {
    sqlQuery()
}
```

Output:

```
[novalagung:belajar-golang $ go run bab53.go
James Bond
Ethan Hunt
novalagung:belajar-golang $ ]
```

#### A.53.4. Membaca 1 Record Data Menggunakan Method `QueryRow()`

Untuk query yang menghasilkan 1 baris record saja, bisa gunakan method `QueryRow()`, dengan metode ini kode menjadi lebih ringkas. Chain dengan method `Scan()` untuk mendapatkan value-nya.

```
func sqlQueryRow() {
    var db, err = connect()
    if err != nil {
        fmt.Println(err.Error())
        return
    }
    defer db.Close()

    var result = student{}
    var id = "E001"
    err = db.
        QueryRow("select name, grade from tb_student where id = ?", id).
        Scan(&result.name, &result.grade)
    if err != nil {
        fmt.Println(err.Error())
        return
    }

    fmt.Printf("name: %s\ngrade: %d\n", result.name, result.grade)
}

func main() {
    sqlQueryRow()
}
```

Dari kode di atas ada statement yang dituliskan cukup unik, chain statement boleh dituliskan dalam beberapa baris, contohnya:

```
err = db.
    QueryRow("select name, grade from tb_student where id = ?", id).
    Scan(&result.name, &result.grade)
```

Sekarang jalankan program. Outputnya akan muncul data record sesuai id.

```
[novalagung:belajar-golang $ go run bab53.go
name: Ethan Hunt
grade: 2
novalagung:belajar-golang $ ]
```

## A.53.5. Eksekusi Query Menggunakan `Prepare()`

Teknik **prepared statement** adalah teknik penulisan query di awal dengan kelebihan bisa di re-use atau digunakan banyak kali untuk eksekusi yang berbeda-beda.

Metode ini bisa digabung dengan `query()` maupun `QueryRow()`. Berikut merupakan contoh penerapannya.

```
func sqlPrepare() {
    db, err := connect()
    if err != nil {
        fmt.Println(err.Error())
        return
    }
    defer db.Close()

    stmt, err := db.Prepare("select name, grade from tb_student where id = ?")
    if err != nil {
        fmt.Println(err.Error())
        return
    }

    var result1 = student{}
    stmt.QueryRow("E001").Scan(&result1.name, &result1.grade)
    fmt.Printf("name: %s\ngrade: %d\n", result1.name, result1.grade)

    var result2 = student{}
    stmt.QueryRow("W001").Scan(&result2.name, &result2.grade)
    fmt.Printf("name: %s\ngrade: %d\n", result2.name, result2.grade)

    var result3 = student{}
    stmt.QueryRow("B001").Scan(&result3.name, &result3.grade)
    fmt.Printf("name: %s\ngrade: %d\n", result3.name, result3.grade)
}

func main() {
    sqlPrepare()
}
```

Method `Prepare()` digunakan untuk deklarasi query, yang mengembalikan objek bertipe `sql.*Stmt`. Dari objek tersebut, dipanggil method `QueryRow()` beberapa kali dengan isi value untuk `id` berbeda-beda untuk tiap pemanggilannya.

```
[novalagung:belajar-golang $ go run bab53.go
name: Ethan Hunt
grade: 2
name: John Wick
grade: 2
name: Jason Bourne
grade: 1
novalagung:belajar-golang $ ]
```

## A.53.6. Insert, Update, & Delete Data Menggunakan `Exec()`

Untuk operasi **insert**, **update**, dan **delete**; dianjurkan untuk tidak menggunakan fungsi `sql.Query()` ataupun `sql.QueryRow()` untuk eksekusinya. Direkomendasikan eksekusi perintah-perintah tersebut lewat fungsi `Exec()`, contohnya seperti pada kode berikut.

```

func sqlExec() {
    db, err := connect()
    if err != nil {
        fmt.Println(err.Error())
        return
    }
    defer db.Close()

    _, err = db.Exec("insert into tb_student values (?, ?, ?, ?)", "G001", "Galahad", 29, 2)
    if err != nil {
        fmt.Println(err.Error())
        return
    }
    fmt.Println("insert success!")

    _, err = db.Exec("update tb_student set age = ? where id = ?", 28, "G001")
    if err != nil {
        fmt.Println(err.Error())
        return
    }
    fmt.Println("update success!")

    _, err = db.Exec("delete from tb_student where id = ?", "G001")
    if err != nil {
        fmt.Println(err.Error())
        return
    }
    fmt.Println("delete success!")
}

func main() {
    sqlExec()
}

```

Teknik prepared statement juga bisa digunakan pada metode ini. Berikut adalah perbandingan eksekusi `Exec()` menggunakan `Prepare()` dan cara biasa.

```

// menggunakan metode prepared statement
stmt, err := db.Prepare("insert into tb_student values (?, ?, ?, ?)")
stmt.Exec("G001", "Galahad", 29, 2)

// menggunakan metode biasa
_, err := db.Exec("insert into tb_student values (?, ?, ?, ?)", "G001", "Galahad", 29, 2)

```

## A.53.7. Koneksi Dengan Engine Database Lain

Karena package `database/sql` merupakan interface generic, maka cara untuk koneksi ke engine database lain (semisal Oracle, Postgres, SQL Server) adalah sama dengan cara koneksi ke MySQL. Cukup dengan meng-import driver yang digunakan, lalu mengganti nama driver pada saat pembuatan koneksi baru.

```
sql.Open(driverName, connectionString)
```

Sebagai contoh saya menggunakan driver `pq` untuk koneksi ke server Postgres, maka connection string-nya:

```
sql.Open("pq", "user=postgres password=secret dbname=test sslmode=disable")
```

Selengkapnya mengenai driver yang tersedia bisa dilihat di <https://github.com/golang/go/wiki/SQLDrivers>.

- [Go MySQL Driver](#), by Julien Schmidt, MPL-2.0 license

## A.54. NoSQL MongoDB

Golang tidak menyediakan interface generic untuk NoSQL, jadi implementasi driver tiap brand NoSQL di Golang bisa berbeda satu dengan lainnya.

Dari sekian banyak teknologi NoSQL yang ada, yang terpilih untuk dibahas di buku ini adalah MongoDB. Dan pada bab ini kita akan belajar cara berkomunikasi dengan MongoDB menggunakan driver [mgo](#).

### A.54.1. Persiapan

Ada beberapa hal yang perlu disiapkan sebelum mulai masuk ke bagian coding.

1. Instal mgo menggunakan `go get` .

```
go get gopkg.in/mgo.v2
```

```
[novalagung:belajar-golang $ go get gopkg.in/mgo.v2
novalagung:belajar-golang $ ]
```

2. Pastikan sudah terinstal MongoDB di komputer anda, dan jangan lupa untuk menjalankan daemon-nya. Jika belum, [download](#) dan install terlebih dahulu.
3. Instal juga MongoDB GUI untuk mempermudah browsing data. Bisa menggunakan [MongoChef](#), [Robomongo](#), atau lainnya.

### A.54.2. Insert Data

Cara insert data lewat mongo tidak terlalu sulit. Kita akan praktikan bagaimana caranya.

Import package yang dibutuhkan, lalu siapkan struct model.

```
package main

import "fmt"
import "gopkg.in/mgo.v2"

type student struct {
    Name string `bson:"name"`
    Grade int    `bson:"Grade"`
}
```

Tag `bson` pada property struct dalam konteks mgo, digunakan sebagai penentu nama field ketika data disimpan kedalam collection. Jika sebuah property tidak memiliki tag `bson`, secara default nama field adalah sama dengan nama property hanya saja lowercase. Untuk customize nama field, gunakan tag `bson` .

Pada contoh di atas, property `Name` ditentukan nama field nya sebagai `name` , dan `Grade` sebagai `Grade` .

Selanjutnya siapkan fungsi untuk membuat session baru.

```
func connect() (*mgo.Session, error) {
    var session, err = mgo.Dial("localhost")
    if err != nil {
        return nil, err
    }
    return session, nil
```

```
}
```

Fungsi `mgo.Dial()` digunakan untuk membuat objek session baru, dengan tipe `*mgo.Session`. Fungsi ini memiliki satu parameter yang harus diisi, yaitu connection string dari server mongo yang akan diakses.

Secara default jenis konsistensi session yang digunakan adalah `mgo.Primary`. Anda bisa mengubahnya lewat method `SetMode()` milik struct `mgo.Session`. Lebih jelasnya silakan merujuk <https://godoc.org/gopkg.in/mgo.v2#Session.SetMode>.

Terkahir buat fungsi insert yang didalamnya berisikan kode untuk insert data ke mongodb, lalu implementasikan di `main`.

```
func insert() {
    var session, err = connect()
    if err != nil {
        fmt.Println("Error!", err.Error())
        return
    }
    defer session.Close()

    var collection = session.DB("belajar_golang").C("student")
    err = collection.Insert(&student{"Wick", 2}, &student{"Ethan", 2})
    if err != nil {
        fmt.Println("Error!", err.Error())
        return
    }

    fmt.Println("Insert success!")
}

func main() {
    insert()
}
```

Session di mgo juga harus di close ketika sudah tidak digunakan, seperti pada instance connection di bab SQL. Statement `defer session.Close()` akan mengakhirkan proses close session dalam fungsi `insert()`.

Struct `mgo.Session` memiliki method `DB()`, digunakan untuk memilih database, dan bisa langsung di chain dengan fungsi `C()` untuk memilih collection.

Setelah mendapatkan instance collection-nya, gunakan method `Insert()` untuk insert data ke database. Method ini memiliki parameter variadic, harus diisi pointer data yang ingin di-insert.

Jalankan program tersebut, lalu cek menggunakan mongo GUI untuk melihat apakah data sudah masuk.

Key	Value	Type
▼ (1) {_id : 562b58c30f04d83ad2}	{ 3 fields }	Document
▀ _id	562b58c30f04d83ad2	ObjectId
▀ name	Wick	String
▀ Grade	2	Int32
▼ (2) {_id : 562b58c30f04d83ad3}	{ 3 fields }	Document
▀ _id	562b58c30f04d83ad3	ObjectId
▀ name	Ethan	String
▀ Grade	2	Int32

### A.54.3. Membaca Data

method `Find()` milik tipe collection `mgo.Collection` digunakan untuk melakukan pembacaan data. Query selectornya dituliskan menggunakan `bson.M` lalu disisipkan sebagai parameter fungsi `Find()`.

Untuk menggunakan `bson.M`, package `gopkg.in/mgo.v2/bson` harus di-import terlebih dahulu.

```
import "gopkg.in/mgo.v2/bson"
```

Setelah itu buat fungsi `find` yang didalamnya terdapat proses baca data dari database.

```
func find() {
    var session, err = connect()
    if err != nil {
        fmt.Println("Error!", err.Error())
        return
    }
    defer session.Close()
    var collection = session.DB("belajar_golang").C("student")

    var result = student{}
    var selector = bson.M{"name": "Wick"}
    err = collection.Find(selector).One(&result)
    if err != nil {
        fmt.Println("Error!", err.Error())
        return
    }

    fmt.Println("Name : ", result.Name)
    fmt.Println("Grade : ", result.Grade)
}

func main() {
    find()
}
```

Variabel `result` diinisialisasi menggunakan struct `student`. Variabel tersebut nantinya digunakan untuk menampung hasil pencarian data.

query selector ditulis dalam tipe `bson.M`. Tipe ini sebenarnya adalah alias dari `map[string]interface{}`.

Selector tersebut kemudian dimasukan sebagai parameter method `Find()`, yang kemudian di chain langsung dengan method `One()` untuk mengambil 1 baris datanya. Pointer variabel `result` disisipkan sebagai parameter method tersebut.

```
[novalagung:belajar-golang $ go run bab54.go
Name : Wick
Grade : 2
novalagung:belajar-golang $ ]
```

## A.54.4. Update Data

Method `Update()` milik struct `mgo.Collection` digunakan untuk update data. Ada 2 parameter yang harus diisi:

1. Parameter pertama adalah query selector data yang ingin di update
2. Parameter kedua adalah data perubahannya

Di bawah ini adalah contoh implementasi method `update()`.

```
func update() {
    var session, err = connect()
    if err != nil {
        fmt.Println("Error!", err.Error())
        return
    }
    defer session.Close()
    var collection = session.DB("belajar_golang").C("student")

    var selector = bson.M{"name": "Wick"}
```

```

var changes = student{"John Wick", 2}
err = collection.Update(selector, changes)
if err != nil {
    fmt.Println("Error!", err.Error())
    return
}

fmt.Println("Update success!")
}

func main() {
    update()
}

```

Jalankan kode di atas, lalu cek lewat Mongo GUI apakah data berubah.

Key	Value	Type
▼ (1) {_id : 562b594f0f04d83c... { 3 fields }	{ 3 fields }	Document
▀ _id	562b594f0f04d83cdd873ad6	ObjectId
▀ name	John Wick	String
▀ Grade	2	Int32
▼ (2) {_id : 562b594f0f04d83c... { 3 fields }	{ 3 fields }	Document
▀ _id	562b594f0f04d83cdd873ad7	ObjectId
▀ name	Ethan	String
▀ Grade	2	Int32

## A.54.5. Menghapus Data

Cara menghapus document pada collection cukup mudah, tinggal gunakan method `Remove()` dengan isi parameter adalah query selector document yang ingin dihapus.

```

func remove() {
    var session, err = connect()
    if err != nil {
        fmt.Println("Error!", err.Error())
        return
    }
    defer session.Close()
    var collection = session.DB("belajar_golang").C("student")

    var selector = bson.M{"name": "John Wick"}
    err = collection.Remove(selector)
    if err != nil {
        fmt.Println("Error!", err.Error())
        return
    }

    fmt.Println("Remove success!")
}

func main() {
    remove()
}

```

2 data yang sebelumnya sudah di-insert kini tinggal satu saja.

Key	Value	Type
▼ (1) {_id : 562b594f0f04d83c... { 3 fields }	{ 3 fields }	Document
▀ _id	562b594f0f04d83cdd873ad7	ObjectId
▀ name	Ethan	String
▀ Grade	2	Int32



## A.55. Unit Test

Golang menyediakan package `testing`, yang berisikan banyak sekali tools untuk keperluan unit testing.

Pada bab ini kita akan belajar mengenai testing, benchmark, dan juga testing menggunakan [testify](#).

### A.55.1. Persiapan

Pertama siapkan terlebih dahulu sebuah struct `Kubus`. Variabel object hasil struct ini nantinya kita gunakan sebagai bahan testing.

```
package main

import "math"

type Kubus struct {
    Sisi float64
}

func (k Kubus) Volume() float64 {
    return math.Pow(k.Sisi, 3)
}

func (k Kubus) Luas() float64 {
    return math.Pow(k.Sisi, 2) * 6
}

func (k Kubus) Keliling() float64 {
    return k.Sisi * 12
}
```

Simpan kode di atas dengan nama `bab55.go`.

### A.55.2. Testing

File untuk keperluan testing dipisah dengan file utama, namanya harus berakhiran `_test.go`, dan package-nya harus sama. Pada bab ini, file utama adalah `bab55.go`, maka file testing harus bernama `bab55_test.go`.

Unit test di Golang dituliskan dalam bentuk fungsi, yang memiliki parameter yang bertipe `*testing.T`, dengan nama fungsi harus diawali kata **Test** (pastikan sudah meng-import package `testing` sebelumnya). Lewat parameter tersebut, kita bisa mengakses method-method untuk keperluan testing.

Pada contoh di bawah ini disiapkan 3 buah fungsi test, yang masing-masing digunakan untuk mengecek apakah hasil kalkulasi volume, luas, dan keliling kubus adalah benar.

```
package main

import "testing"

var (
    kubus         Kubus = Kubus{4}
    volumeSeharusnya float64 = 64
    luasSeharusnya  float64 = 96
    kelilingSeharusnya float64 = 48
)

func TestHitungVolume(t *testing.T) {
```

```
t.Logf("Volume : %.2f", kubus.Volume())

if kubus.Volume() != volumeSeharusnya {
    t.Errorf("SALAH! harusnya %.2f", volumeSeharusnya)
}

func TestHitungLuas(t *testing.T) {
    t.Logf("Luas : %.2f", kubus.Luas())

    if kubus.Luas() != luasSeharusnya {
        t.Errorf("SALAH! harusnya %.2f", luasSeharusnya)
    }
}

func TestHitungKeliling(t *testing.T) {
    t.Logf("Keliling : %.2f", kubus.Keliling())

    if kubus.Keliling() != kelilingSeharusnya {
        t.Errorf("SALAH! harusnya %.2f", kelilingSeharusnya)
    }
}
```

Method `t.Logf()` digunakan untuk memunculkan log. Method ini equivalen dengan `fmt.Printf()`.

Method `Errorf()` digunakan untuk memunculkan log dengan diikuti keterangan bahwa terjadi **fail** pada saat testing.

Cara eksekusi testing adalah menggunakan command `go test`. Karena struct yang diuji berada dalam file `bab55.go`, maka pada saat eksekusi test menggunakan `go test`, nama file `bab55_test.go` dan `bab55.go` perlu dituliskan sebagai argument.

Argument `-v` atau verbose digunakan menampilkan semua output log pada saat pengujian.

Jalankan aplikasi seperti gambar dibawah ini, terlihat bahwa tidak ada test yang fail.

```
[novalagung:belajar-golang $ go test bab55.go bab55_test.go -v
==== RUN TestHitungVolume
--- PASS: TestHitungVolume (0.00s)
    bab55_test.go:13: Volume : 64.00
==== RUN TestHitungLuas
--- PASS: TestHitungLuas (0.00s)
    bab55_test.go:21: Luas : 96.00
==== RUN TestHitungKeliling
--- PASS: TestHitungKeliling (0.00s)
    bab55_test.go:29: Keliling : 48.00
PASS
ok      command-line-arguments 0.005s
novalagung:belajar-golang $ ]
```

OK, selanjutnya coba ubah rumus kalkulasi method `Keliling()`. Tujuan dari pengubahan ini adalah untuk mengetahui bagaimana penanda fail muncul ketika ada test yang gagal.

```
func (k Kubus) Keliling() float64 {
    return k.Sisi * 15
}
```

Setelah itu jalankan lagi test.

```
[novalagung:belajar-golang $ go test bab55.go bab55_test.go -v
    === RUN TestHitungVolume
    --- PASS: TestHitungVolume (0.00s)
        bab55_test.go:13: Volume : 64.00
    === RUN TestHitungLuas
    --- PASS: TestHitungLuas (0.00s)
        bab55_test.go:21: Luas : 96.00
    === RUN TestHitungKeliling
    --- FAIL: TestHitungKeliling (0.00s)
        bab55_test.go:29: Keliling : 60.00
        bab55_test.go:32: SALAH! harusnya 48.00
FAIL
exit status 1
FAIL    command-line-arguments  0.005s
novalagung:belajar-golang $ ]
```

### A.55.3. Method Test

Table berikut berisikan method standar testing yang bisa digunakan di Golang.

Method	Kegunaan
<code>Log()</code>	Menampilkan log
<code>Logf()</code>	Menampilkan log menggunakan format
<code>Fail()</code>	Menandakan terjadi <code>Fail()</code> dan proses testing fungsi tetap diteruskan
<code>FailNow()</code>	Menandakan terjadi <code>Fail()</code> dan proses testing fungsi dihentikan
<code>Failed()</code>	Menampilkan laporan fail
<code>Error()</code>	<code>Log()</code> diikuti dengan <code>Fail()</code>
<code>Errorf()</code>	<code>Logf()</code> diikuti dengan <code>Fail()</code>
<code>Fatal()</code>	<code>Log()</code> diikuti dengan <code>failNow()</code>
<code>Fatalf()</code>	<code>Logf()</code> diikuti dengan <code>failNow()</code>
<code>Skip()</code>	<code>Log()</code> diikuti dengan <code>SkipNow()</code>
<code>Skipf()</code>	<code>Logf()</code> diikuti dengan <code>SkipNow()</code>
<code>SkipNow()</code>	Menghentikan proses testing fungsi, dilanjutkan ke testing fungsi setelahnya
<code>Skipped()</code>	Menampilkan laporan skip
<code>Parallel()</code>	Menge-set bahwa eksekusi testing adalah parallel

### A.55.4. Benchmark

Package `testing` selain berisikan tools untuk testing juga berisikan tools untuk benchmarking. Cara pembuatan benchmark sendiri cukup mudah yaitu dengan membuat fungsi yang namanya diawali dengan **Benchmark** dan parameternya bertipe `*testing.B`.

Sebagai contoh, kita akan mengetes performa perhitungan luas kubus. Siapkan fungsi dengan nama `BenchmarkHitungLuas()` dengan isi adalah kode berikut.

```
func BenchmarkHitungLuas(b *testing.B) {
    for i := 0; i < b.N; i++ {
        kubus.Luas()
    }
}
```

Jalankan test menggunakan argument `-bench=.`, argumen ini digunakan untuk menandai bahwa selain testing terdapat juga benchmark yang perlu diuji.

```
[novalagung:belajar-golang $ go test bab55.go bab55_test.go -bench=.]
PASS
BenchmarkHitungLuas      30000000          51.4 ns/op
ok   command-line-arguments 1.598s
novalagung:belajar-golang $ ]
```

Arti dari `30000000 51.1 ns/op` adalah, fungsi di atas di-test sebanyak **30 juta** kali, hasilnya membutuhkan waktu rata-rata **51 nano detik** untuk run satu fungsi.

## A.55.5. Testing Menggunakan `testify`

Package **testify** berisikan banyak sekali tools yang bisa dimanfaatkan untuk keperluan testing di Golang.

Testify bisa di-download pada [github.com/stretchr/testify](https://github.com/stretchr/testify) menggunakan `go get .`

Didalam testify terdapat 5 package dengan kegunaan berbeda-beda satu dengan lainnya. Detailnya bisa dilihat pada tabel berikut.

Package	Kegunaan
assert	Berisikan tools standar untuk testing
http	Berisikan tools untuk keperluan testing http
mock	Berisikan tools untuk mocking object
require	Sama seperti assert, hanya saja jika terjadi fail pada saat test akan menghentikan eksekusi program
suite	Berisikan tools testing yang berhubungan dengan struct dan method

Pada bab ini akan kita contohkan bagaimana penggunaan package assert. Silakan perhatikan contoh berikut.

```
import "github.com/stretchr/testify/assert"

...

func TestHitungVolume(t *testing.T) {
    assert.Equal(t, kubus.Volume(), volumeSeharusnya, "perhitungan volume salah")
}

func TestHitungLuas(t *testing.T) {
    assert.Equal(t, kubus.Luas(), luasSeharusnya, "perhitungan luas salah")
}

func TestHitungKeliling(t *testing.T) {
    assert.Equal(t, kubus.Keliling(), kelilingSeharusnya, "perhitungan keliling salah")
}
```

Fungsi `assert.Equal()` digunakan untuk uji perbandingan. Parameter ke-2 dibandingkan nilainya dengan parameter ke-3. Jika tidak sama, maka pesan parameter ke-3 akan dimunculkan.

```
[novalagung:belajar-golang $ go test bab55.go bab55_test.go -v
    === RUN TestHitungVolume
    --- PASS: TestHitungVolume (0.00s)
    === RUN TestHitungLuas
    --- PASS: TestHitungLuas (0.00s)
    === RUN TestHitungKeliling
    --- FAIL: TestHitungKeliling (0.00s)
            Error Trace:    bab55_test.go:24
            Error:           Not equal: 60 (expected)
                                != 48 (actual)
            Messages:        perhitungan keliling salah

FAIL
exit status 1
FAIL    command-line-arguments  0.008s
novalagung:belajar-golang $ ]
```

- 
- [Testify](#), by "Stretchr, Inc"

## A.56. WaitGroup

Sebelumnya kita telah belajar banyak mengenai channel, yang fungsi utama-nya adalah untuk sharing data antar goroutine. Selain untuk komunikasi data, channel secara tidak langsung bisa dimanfaatkan untuk mengontrol goroutine.

Golang menyediakan package `sync`, berisi cukup banyak API untuk handle masalah multiprocessing (goroutine), salah satunya diantaranya adalah yang kita bahas di bab ini, yaitu `sync.WaitGroup`.

Kegunaan `sync.WaitGroup` adalah untuk sinkronisasi goroutine. Berbeda dengan channel, `sync.WaitGroup` memang didesain khusus untuk maintain goroutine, penggunaannya lebih mudah dan lebih efektif dibanding channel.

Sebenarnya kurang pas membandingkan `sync.WaitGroup` dan channel, karena fungsi utama dari keduanya adalah berbeda. Channel untuk keperluan sharing data antar goroutine, sedangkan `sync.WaitGroup` untuk sinkronisasi goroutine.

### A.56.1. Penerapan `sync.WaitGroup`

`sync.WaitGroup` digunakan untuk menunggu goroutine. Cara penggunaannya sangat mudah, tinggal masukan jumlah goroutine yang dieksekusi, sebagai parameter method `Add()` object cetakan `sync.WaitGroup`. Dan pada akhir tiap-tiap goroutine, panggil method `Done()`. Juga, pada baris kode setelah eksekusi goroutine, panggil method `Wait()`.

Agar lebih jelas, silakan coba kode berikut.

```
package main

import "sync"
import "runtime"
import "fmt"

func doPrint(wg *sync.WaitGroup, message string) {
    defer wg.Done()
    fmt.Println(message)
}

func main() {
    runtime.GOMAXPROCS(2)

    var wg sync.WaitGroup

    for i := 0; i < 5; i++ {
        var data = fmt.Sprintf("data %d", i)

        wg.Add(1)
        go doPrint(&wg, data)
    }

    wg.Wait()
}
```

Kode di atas merupakan contoh penerapan `sync.WaitGroup` untuk pengelolahan goroutine. Fungsi `doPrint()` akan dijalankan sebagai goroutine, dengan tugas menampilkan isi variabel `message`.

Variabel `wg` disiapkan bertipe `sync.WaitGroup`, dibuat untuk sinkronisasi goroutines yang dijalankan.

Di tiap perulangan statement `wg.Add(1)` dipanggil. Kode tersebut akan memberikan informasi kepada `wg` bahwa jumlah goroutine yang sedang diproses ditambah 1 (karena dipanggil 5 kali, maka `wg` akan sadar bahwa terdapat 5 buah goroutine sedang berjalan).

Di baris selanjutnya, fungsi `doPrint()` dieksekusi sebagai goroutine. Didalam fungsi tersebut, sebuah method bernama `Done()` dipanggil. Method ini digunakan untuk memberikan informasi kepada `wg` bahwa goroutine dimana method itu dipanggil sudah selesai. Sejumlah 5 buah goroutine dijalankan, maka method tersebut harus dipanggil 5 kali.

Statement `wg.Wait()` bersifat blocking, proses eksekusi program tidak akan diteruskan ke baris selanjutnya, sebelum sejumlah 5 goroutine selesai. Jika `Add(1)` dipanggil 5 kali, maka `Done()` juga harus dipanggil 5 kali.

Output program di atas.

```
[novalagung:belajar-golang $ go run bab56.go
data 4
data 2
data 3
data 0
data 1
[novalagung:belajar-golang $ go run bab56.go
data 4
data 1
data 2
data 3
data 0
novalagung:belajar-golang $ ]
```

## A.56.2. Perbedaan WaitGroup Dengan Channel

Beberapa perbedaan antara channel dan `sync.WaitGroup` :

- Channel tergantung kepada goroutine tertentu dalam penggunaannya, tidak seperti `sync.WaitGroup` yang dia tidak perlu tahu goroutine mana saja yang dijalankan, cukup tahu jumlah goroutine yang harus selesai
- Penerapan `sync.WaitGroup` lebih mudah dibanding channel
- Kegunaan utama channel adalah untuk komunikasi data antar goroutine. Sifatnya yang blocking bisa kita manfaatkan untuk manage goroutine; sedangkan WaitGroup khusus digunakan untuk sinkronisasi goroutine
- Performa `sync.WaitGroup` lebih baik dibanding channel, sumber: <https://groups.google.com/forum/#topic/golang-nuts/wphCEk9yLhc>

Kombinasi yang tepat antara `sync.WaitGroup` dan channel sangat penting, dibutuhkan untuk handle proses concurrent agar bisa maksimal.

## A.57. Mutex

Sebelum kita membahas mengenai apa itu **mutex**? ada baiknya untuk mempelajari terlebih dahulu apa itu **race condition**, karena kedua konsep ini berhubungan erat satu sama lain.

Race condition adalah kondisi dimana lebih dari satu goroutine, mengakses data yang sama pada waktu yang bersamaan (benar-benar bersamaan). Ketika hal ini terjadi, nilai data tersebut akan menjadi kacau. Dalam **concurrency programming** situasi seperti ini ini sering terjadi.

Mutex melakukan pengubahan level akses sebuah data menjadi eksklusif, menjadikan data tersebut hanya dapat dikonsumsi (read / write) oleh satu buah goroutine saja. Ketika terjadi race condition, maka hanya goroutine yang beruntung saja yang bisa mengakses data tersebut. Goroutine lain (yang waktu running nya kebetulan bersamaan) akan dipaksa untuk menunggu, hingga goroutine yang sedang memanfaatkan data tersebut selesai.

Golang menyediakan `sync.Mutex` yang bisa dimanfaatkan untuk keperluan **lock** dan **unlock** data. Pada bab ini kita akan membahas mengenai race condition, dan menanggulanginya menggunakan mutex.

### A.57.1. Persiapan

Pertama siapkan struct baru bernama `counter`, dengan isi satu buah property `val` bertipe `int`. Property ini nantinya dikonsumsi dan diolah oleh banyak goroutine.

Lalu buat beberapa method struct `counter`.

1. Method `Add()`, untuk increment nilai.
2. Method `Value()`, untuk mengembalikan nilai.

```
package main

import (
    "fmt"
    "runtime"
    "sync"
)

type counter struct {
    val int
}

func (c *counter) Add(x int) {
    c.val++
}

func (c *counter) Value() (x int) {
    return c.val
}
```

Kode di atas kita gunakan sebagai template contoh source code yang ada pada bab ini.

### A.57.2. Contoh Race Condition

Program berikut merupakan contoh program yang didalamnya memungkinkan terjadi race condition atau kondisi goroutine balapan.

Pastikan jumlah core prosesor komputer anda adalah lebih dari satu. Karena contoh pada bab ini hanya akan berjalan sesuai harapan jika `GOMAXPROCS > 1`.

```
func main() {
    runtime.GOMAXPROCS(2)

    var wg sync.WaitGroup
    var meter counter

    for i := 0; i < 1000; i++ {
        wg.Add(1)

        go func() {
            for j := 0; j < 1000; j++ {
                meter.Add(1)
            }
        }

        wg.Done()
    }

    wg.Wait()
    fmt.Println(meter.Value())
}
```

Pada kode diatas, disiapkan sebuah instance `sync.WaitGroup` bernama `wg`, dan variabel object `meter` bertipe `counter` (nilai property `val` defaultnya adalah `0`).

Setelahnya dijalankan perulangan sebanyak 1000 kali, yang ditiap perulangannya dijalankan sebuah goroutine baru. Didalam goroutine tersebut, terdapat perulangan lagi, sebanyak 1000 kali. Dalam perulangan tersebut nilai property `val` dinaikkan sebanyak 1 lewat method `Add()`.

Dengan demikian, ekspektasi nilai akhir `meter.val` harusnya adalah 1000000.

Di akhir, `wg.Wait()` dipanggil, dan nilai variabel counter `meter` diambil lewat `meter.Value()` untuk kemudian ditampilkan.

Jalankan program, lihat hasilnya.

```
[novalagung:belajar-golang $ go run bab57.go
754541
[novalagung:belajar-golang $ go run bab57.go
753642
[novalagung:belajar-golang $ go run bab57.go
729100
novalagung:belajar-golang $ ]]
```

Nilai `meter.val` tidak genap 1000000? kenapa bisa begitu? Padahal seharusnya tidak ada masalah dalam kode yang kita tulis di atas.

Inilah yang disebut dengan race condition, data yang diakses bersamaan dalam 1 waktu menjadi kacau.

### A.57.3. Deteksi Race Condition Menggunakan Golang Race Detector

Golang menyediakan fitur untuk [deteksi race condition](#). Cara penggunaannya adalah dengan menambahkan flag `-race` pada saat eksekusi aplikasi.

```
[novalagung:belajar-golang $ go run -race bab57.go
=====
WARNING: DATA RACE
Read by goroutine 7:
 main.main.func1()
 /Users/novalagung/Documents/go/src/belajar-golang/bab57.go:30 +0x46

Previous write by goroutine 6:
 main.main.func1()
 /Users/novalagung/Documents/go/src/belajar-golang/bab57.go:30 +0x5a

Goroutine 7 (running) created at:
 main.main()
 /Users/novalagung/Documents/go/src/belajar-golang/bab57.go:34 +0xe8

Goroutine 6 (finished) created at:
 main.main()
 /Users/novalagung/Documents/go/src/belajar-golang/bab57.go:34 +0xe8
=====
679625
Found 1 data race(s)
exit status 66
novalagung:belajar-golang $ ]
```

Terlihat pada gambar diatas, ada pesan memberitahu terdapat kemungkinan data race pada program yang kita jalankan.

#### A.57.4. Penerapan sync.Mutex

Sekarang kita tahu bahwa program di atas menghasilkan bug, ada kemungkinan data race didalamnya. Untuk mengatasi masalah ini ada beberapa cara yang bisa digunakan, dan disini kita akan menggunakan `sync.Mutex`.

Ubah kode di atas, embed struct `sync.Mutex` kedalam struct `counter`, agar lewat objek cetakan `counter` kita bisa melakukan lock dan unlock dengan mudah. Tambahkan method `Lock()` dan `unlock()` didalam method `Add()`.

```
type counter struct {
    sync.Mutex
    val int
}

func (c *counter) Add(x int) {
    c.Lock()
    c.val++
    c.Unlock()
}

func (c *counter) Value() (x int) {
    return c.val
}
```

Method `Lock()` digunakan untuk menandai bahwa semua operasi pada baris setelah kode tersebut adalah bersifat eksklusif. Hanya ada satu buah goroutine yang bisa melakukannya dalam satu waktu. Jika ada banyak goroutine yang eksekusinya bersamaan, harus antri.

Pada kode di atas terdapat kode untuk increment nilai `meter.val`. Maka property tersebut hanya bisa diakses oleh satu goroutine saja.

Method `Unlock()` akan membuka kembali akses operasi ke property/variabel yang di lock, proses mutual exclusion terjadi diantara method `Lock()` dan `Unlock()`.

Di contoh di atas, pada saat bagian pengambilan nilai, mutex tidak dipasang, karena kebetulan pengambilan nilai terjadi setelah semua goroutine selesai. Data Race bisa terjadi saat pengubahan maupun pengambilan data, jadi penggunaan mutex harus disesuaikan dengan kasus.

Coba jalankan program, dan lihat hasilnya.

```
[novalagung:belajar-golang $ go run bab57.go
1000000
[novalagung:belajar-golang $ go run bab57.go
1000000
[novalagung:belajar-golang $ go run bab57.go
1000000
[novalagung:belajar-golang $ ]
```

Penggunaan `sync.Mutex` yang dianjurkan adalah dengan cara langsung di embed ke struct dimana proses yang memungkinkan race condition berada. Kita bisa saja menggunakan mutex dengan cara biasa, berikut adalah contohnya.

```
func (c *counter) Add(x int) {
    c.val++
}

func (c *counter) Value() (x int) {
    return c.val
}

func main() {
    runtime.GOMAXPROCS(2)

    var wg sync.WaitGroup
    var mtx sync.Mutex
    var meter counter

    for i := 0; i < 1000; i++ {
        wg.Add(1)

        go func() {
            for j := 0; j < 1000; j++ {
                mtx.Lock()
                meter.Add(1)
                mtx.Unlock()
            }
        }

        wg.Done()
    }()
}

wg.Wait()
fmt.Println(meter.Value())
}
```

## A.58. Go Vendoring

Pada bagian ini kita akan belajar cara pemanfaatan vendoring untuk mempermudah manajemen package atau dependency dalam project Go.

Pada Bab A.3. GOPATH Dan Workspace sudah disinggung bahwa project Go harus ditempatkan didalam workspace, lebih spesifiknya dalam folder `$GOPATH/src/`. Aturan ini cukup memicu perdebatan di komunitas, karena menghasilkan efek negatif terhadap beberapa hal, yang salah satunya adalah: dependency management yang dirasa susah.

### A.58.1. Perbandingan Project Yang Menerapkan Vendoring vs Tidak

Penulis coba contohkan dengan sebuah kasus untuk mempermudah pembaca memahami permasalahan yang ada dalam dependency management di Go.

Dimisalkan, ada dua buah projek yang sedang di-develop, `project-one` dan `project-two`. Keduanya depend terhadap salah satu 3rd party library yg sama, `gubrak`. Di dalam `project-one`, versi gubrak yang digunakan adalah `v0.9.1-alpha`, sedangkan di `project-two` versi `v1.0.0` digunakan. Pada `project-one` versi yang digunakan cukup tua karena proses pengembangannya sudah agak lama, dan aplikasinya sendiri sudah stabil, jika di upgrade paksa ke gubrak versi `v1.0.0` kemungkinan besar terjadi error dan panic.

Kedua projek tersebut pastinya akan lookup gubrak ke direktori yang sama, yaitu `$GOPATH/src/github.com/novalagung/gubrak`. Efeknya, ketika sedang bekerja pada `project-one`, harus dipastikan current revision pada repository gubrak di lokal adalah sesuai dengan versi `v1.0.0`. Dan, ketika mengerjakan `project-two` maka current revision gubrak harus sesuai dengan versi `v0.9.1-alpha`. Repot sekali bukan?

Setelah beberapa waktu, akhirnya `go1.6` rilis, dengan membawa kabar baik, yaitu rilisnya fasilitas baru **vendoring**. Vendoring ini berguna untuk men-centralize packages atau dependencies atau 3rd party libraries yang digunakan dalam spesifik project.

Penggunaannya sendiri sangat mudah, cukup tempatkan 3rd party library ke-dalam folder `vendor`, yang berada di dalam masing-masing project. By default go akan memprioritaskan lookup pada folder `vendor`.

Folder `vendor` ini kegunaannya sama seperti folder `node_modules` dalam project javascript.

Kita kembali ke contoh, library gubrak dipindahkan ke folder `vendor` dalam `project-one`, maka struktur project tersebut plus vendoring menjadi seperti berikut.

```
$GOPATH/src/project-one
$GOPATH/src/project-one/vendor/github.com/novalagung/gubrak
```

Source code library gubrak dalam folder `vendor` harus ditempatkan sesuai dengan strukturnya.

```
$ tree .
.
└── main.go
└── vendor
    └── github.com
        └── novalagung
            └── gubrak
                ├── date.go
                ├── is.go
                └── ...

```

```
4 directories, N files
```

Isi `project-one/main.go` sendiri cukup sederhana, sebuah program kecil yang menampilkan angka random dengan range 10-20.

```
package main

import (
    "fmt"
    "github.com/novalagung/gubrak"
)

func main() {
    fmt.Println(gubrak.RandomInt(10, 20))
}
```

Ketika di build, dengan flag `-v` terlihat perbedaan antara projek yang menerapkan vendoring maupun yang tidak.

- Untuk yang tidak menggunakan vendor folder, maka akan lookup ke folder library yang digunakan, di `$GOPATH`.
- Untuk project yang menerapkan vendoring, maka tetap lookup juga, tetapi dalam sub folder `vendor` yang ada di dalam projek tersebut.

The screenshot shows two terminal windows side-by-side. The top terminal window is labeled 'without vendoring' and displays the command `novalagung:project-one $ GODEBUG=gocacheverify=1 go build -v` followed by the output `github.com/novalagung/gubrak`. The bottom terminal window is labeled 'with vendoring' and displays the same command and output, but the package name is resolved to its full path within the vendor directory: `project-one/vendor/github.com/novalagung/gubrak`.

## A.58.2. Manajemen Dependencies Dalam Folder `vendor`

Cara menambahkan dependency ke folder `vendor` bisa dengan cara copy-paste, yang tetapi jelasnya adalah tidak praktis (dan bisa menimbulkan masalah). Cara yang lebih benar adalah dengan menggunakan package management tools.

Di go, ada sangat banyak sekali package management tools. Sangat. Banyak. Sekali. Silakan cek di [PackageManagementTools](#) untuk lebih detailnya.

Pembaca bebas memilih untuk menggunakan package management tools yang mana. Namun di buku ini, **Dep** akan kita bahas. Dep sendiri merupakan official package management tools untuk Go dari Go Team.

Pembahasan mengenai Dep ada di bab selanjutnya.

## A.58. Dep - Go Dependency Management Tool

Dep adalah Go Official Package Management Tools, diperkenalkan pada go versi `go1.9`. Pada bab ini kita akan belajar penggunaannya.

Dengan menggunakan Dep, maka semua 3rd party libraries yang dipakai dalam suatu project akan dicatat dan ditempatkan di dalam sub folder `vendor` pada project tersebut.

### A.58.1. Instalasi Dep

Silakan ikuti petunjuk di [Dep Installation](#), ada beberapa cara yang bisa dipilih untuk meng-install Dep.

### A.58.2. Inisialisasi Dep Pada Project

Buat project baru, isi dengan kode sederhana untuk menampilkan angka random (sama seperti pada bab sebelumnya). Library yang kita gunakan adalah [gubrak](#).

```
package main

import (
    "fmt"
    "github.com/novalagung/gubrak"
)

func main() {
    fmt.Println(gubrak.RandomInt(10, 20))
}
```

Jalankan command `dep init` untuk meng-inisialisasi project agar ditandai sebagai project yang menggunakan package manager Dep. Command tersebut menghasilkan 3 buah file/folder baru.

- File `Gopkg.toml`, berisikan metada semua dependencies yang digunakan dalam project.
- File `Gopkg.lock`, isinya mirip seperti `Gopkg.toml` hanya saja lebih mendetail informasi tiap-tiap dependency yang disimpan dalam file ini.
- Folder `vendor`, akan berisi source code dari pada 3rd party yang digunakan.

Eksekusi command `dep init` akan secara otomatis diikuti dengan eksekusi `dep ensure`. Command `dep ensure` sendiri merupakan command paling penting dalam Dep, gunanya untuk sinkronisasi dan memastikan bahwa semua 3rd party libraries digunakan dalam project metadata-nya ter-mapping dengan baik dan benar dalam `Gopkg.*`, dan source code 3rd party sendiri ada dalam `vendor`.

```
$ tree -L 1 .
.
├── Gopkg.lock
├── Gopkg.toml
├── main.go
└── vendor

1 directory, 3 files
```

Selesai. Sesederhana ini. Cukup mudah bukan?

### A.58.3. Add & Update Dependency

Gunakan `dep ensure -add <library>` untuk menambahkan library. Gunakan `dep ensure -update <library>` untuk meng-update library yang sudah tercatat.

Penggunaan dua command tersebut jelasnya akan mengubah informasi dalam file `Gopkg.*`. Pastikan bahwa library yang didaftarkan adalah yang memang digunakan di project. Jika ragu, maka hapus saja folder `vendor`, lalu eksekusi command `dep ensure` untuk sinkronisasi ulang.

Contoh command update:

```
$ dep ensure -update github.com/novalagung/gubrak
```

## A.60. Go Modules

Pada bagian ini kita akan belajar cara pemanfaatan Go Modules untuk manajemen project dan dependency. Go Modules merupakan bagian dari **Go Toolchain** yang sekarang sedang dikembangkan.

### A.60.1. The Infamous `$GOPATH`

Sebelumnya sudah di bahas dalam [Bab A.3. GOPATH Dan Workspace](#) bahwa project Go harus ditempatkan didalam workspace, lebih spesifiknya dalam folder `$GOPATH/src/`. Sudah dibahas juga bahwa ada baiknya untuk men-centralize 3rd party library yang digunakan per project dengan memanfaatkan [vendoring](#) dan [package management tools: Dep](#).

Sebenarnya masih ada satu masalah yang masih belum terselesaikan yang berhubungan dengan manajemen project dan dependency di Go. Yaitu `$GOPATH`. Ya, `$GOPATH` adalah **masalah besar** untuk beberapa kasus.

Perlu diketahui, meskipun sebuah project sudah menerapkan vendoring (semua 3rd party sudah ada di sub folder `vendor`), project itu sendiri masih harus ditempatkan dalam workspace!

Silakan lihat gambar di bawah ini untuk pembuktian. Di dalam project `chapter-A.60-go-modules` terdapat package `models` yang harus di-import. Karena project tidak ditempatkan dalam workspace, eksekusinya gagal.



```

chapter-A.60-go-modules $ tree .
.
├── main.go
└── models
    └── user.go

1 directory, 2 files
chapter-A.60-go-modules $ cat main.go
package main

import (
    "fmt"

    "chapter-A.60-go-modules/models"
)

func main() {
    user := models.User{"u001", "Noval"}
    fmt.Println(user)
}
chapter-A.60-go-modules $ go run main.go
main.go:6:2: cannot find package "chapter-A.60-go-modules/models" in any of:
    /usr/local/opt/go/libexec/src/chapter-A.60-go-modules/models (from $GOROOT)
    /Volumes/Nightwing/go/src/chapter-A.60-go-modules/models (from $GOPATH)
chapter-A.60-go-modules $

```

Jika dalam satu project hanya ada satu package, `main` saja, maka meski projek tidak ditempatkan dalam workspace, tidak akan terjadi error. Tapi jelasnya tidak mungkin dalam real world development sebuah project hanya memiliki satu package saja. Brutal sekali kalau misal ada

Bisa dilihat dari stack trace error pada gambar di atas, proses pencarian package `models` dilakukan hanya pada `$GOROOT` dan `$GOPATH` saja, pencarian dalam folder project itu sendiri tidak dilakukan. Ini merupakan pengingat halus bahwa project ini harus ditempatkan dalam workspace, agar package `models`-nya bisa dikenali.

Masalah bukan?

### A.60.2. Go Modules

Setelah cukup lama menunggu, akhirnya pada rilisnya `go1.11` dikenalkanlah sebuah fasilitas baru bernama **Go Modules** (experimental). Tujuan diciptakannya Go Modules ini adalah untuk mengatasi masalah ketergantungan sebuah project dengan `$GOPATH`. Jadi dengan mengimplementasikan Go Modules, sebuah project tidak harus berada dalam `$GOPATH`.

Cara penerapan Go Modules cukup mudah. Inisialisasi project sebagai module lewat command `go mod init <module-name>`.

```
i  OK      chapter-A.60-go-modules — bash — NVL — 87x21
[novalagung:chapter-A.60-go-modules $ tree .
.
├── main.go
└── models
    └── user.go

1 directory, 2 files
[novalagung:chapter-A.60-go-modules $ go mod init chapter-A.60-go-modules
go: creating new go.mod: module chapter-A.60-go-modules
[novalagung:chapter-A.60-go-modules $ tree .
.
├── go.mod
├── main.go
└── models
    └── user.go

1 directory, 3 files
[novalagung:chapter-A.60-go-modules $ go run main.go
{u001 Noval}
novalagung:chapter-A.60-go-modules $ ]
```

Bisa dilihat di gambar, bahwa setelah project di-inisialisasi sebagai module, proses running aplikasi menjadi sukses.

### A.60.3. Penjelasan `go mod`

Command `go mod init` menghasilkan sebuah file bernama `go.mod`, yang isinya adalah konfigurasi module.

Ok, mungkin akan muncul beberapa pertanyaan. Salah satunya adalah kenapa command-nya harus `go mod init chapter-A.60-go-modules`, kenapa tidak cukup hanya `go mod init` saja? Hal ini karena mulai awal, projek ini sudah tidak didalam `$GOPATH`. Argument `chapter-A.60-go-modules` setelah command inisialisasi berguna untuk menentukan package path dari project, hal ini penting untuk keperluan import package lain yang merupakan sub folder project.

Agar lebih mudah untuk dipahami, silakan lihat contoh berikut.

```
i  OK      chapter-A.60-go-modules — bash — NVL — 64x25
[novalagung:chapter-A.60-go-modules $ tree .
.
├── main.go
└── models
    └── user.go

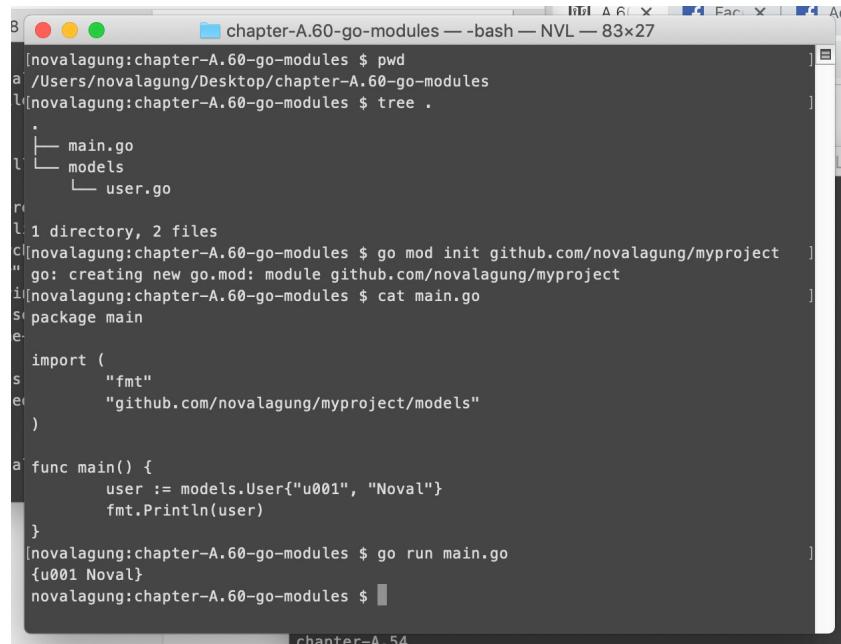
1 directory, 2 files
[novalagung:chapter-A.60-go-modules $ cat main.go
package main
import (
    "fmt"
    "myproject/models"
)
func main() {
    user := models.User{"u001", "Noval"}
    fmt.Println(user)
}

[novalagung:chapter-A.60-go-modules $ go mod init myproject
go: creating new go.mod: module myproject
[novalagung:chapter-A.60-go-modules $ go run main.go
{u001 Noval}
novalagung:chapter-A.60-go-modules $ ]
```

Coba perhatikan gambar di atas. Nama folder adalah `chapter-A.60-go-modules`, tetapi pada saat import `models`, package path yang digunakan adalah `myproject/models`. Harusnya ini error kan? Tetapi tidak, *hmmmm*.

Dengan menggunakan Go Modules kita bisa menentukan base package path suatu project, tidak harus sama dengan nama folder-nya. Pada contoh ini nama folder adalah `chapter-A.60-go-modules` tapi base package path adalah `myproject`.

Agar lebih jelas lagi, lanjut ke contoh ke-dua.



```

[novalagung:chapter-A.60-go-modules $ pwd
a /Users/novalagung/Desktop/chapter-A.60-go-modules
l[novalagung:chapter-A.60-go-modules $ tree .
.
└── main.go
    └── models
        └── user.go
r
l. 1 directory, 2 files
c[novalagung:chapter-A.60-go-modules $ go mod init github.com/novalagung/myproject
" go: creating new go.mod: module github.com/novalagung/myproject
i[novalagung:chapter-A.60-go-modules $ cat main.go
s package main
e
    import (
s         "fmt"
e         "github.com/novalagung/myproject/models"
    )
a func main() {
        user := models.User{"u001", "Noval"}
        fmt.Println(user)
    }
[novalagung:chapter-A.60-go-modules $ go run main.go
{u001 Noval}
novalagung:chapter-A.60-go-modules $

```

Pada contoh ini, nama module di-set panjang, `github.com/novalagung/myproject`. Dengan nama folder masih tetap sama. Hasilnya aplikasi tetap berjalan lancar.

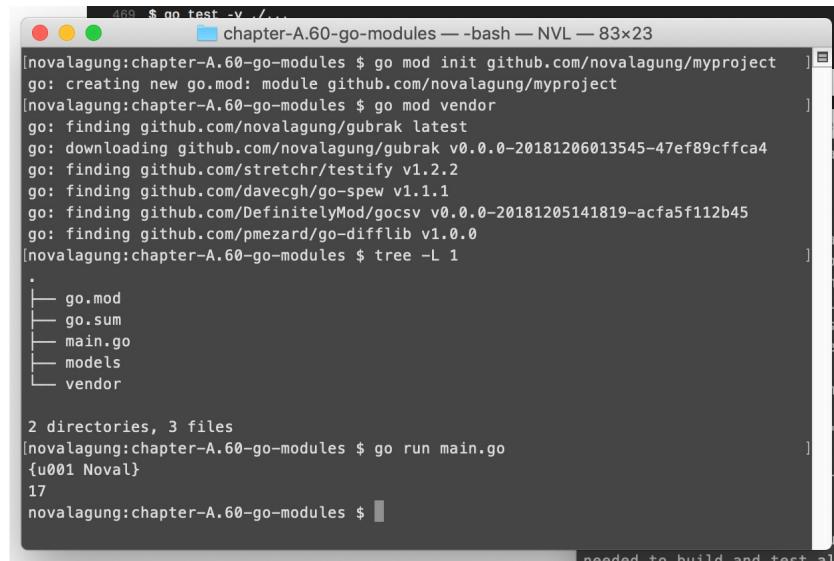
Nama package path bisa di set dalam bentuk arbitrary path seperti pada gambar. Sangat membantu bukan sekali, tidak perlu membuat nested folder terlebih dahulu seperti yang biasa dilakukan dalam `$GOPATH`.

## A.60.4. Vendoring Dengan Go Modules

Vendoring, seperti yang kita tau berguna untuk men-centralize 3rd party libraries. Sedangkan dari yang sudah dipelajari, kita bisa membuat sebuah project untuk tidak tergantung dengan `$GOPATH` lewat Go Modules.

Sebenarnya, Go Modules, selain untuk membuat sebuah project tidak tergantung dengan `$GOPATH`, adalah juga berguna untuk manajemen dependencies project. Jadi dengan Go Modules tidak perlu menggunakan `dep`, karena di dalam Go Modules sudah include kapabilitas yang sama dengan `dep` yaitu untuk me-manage dependencies.

Cara untuk meng-enable vendoring dengan Go Modules, adalah lewat command `go mod vendor`.



```
469 $ go test -v ./...
chapter-A.60-go-modules — bash — NVL — 83x23
[novalagung:chapter-A.60-go-modules $ go mod init github.com/novalagung/myproject
go: creating new go.mod: module github.com/novalagung/myproject
[novalagung:chapter-A.60-go-modules $ go mod vendor
go: finding github.com/novalagung/gubrak latest
go: downloading github.com/novalagung/gubrak v0.0.0-20181206013545-47ef89cffca4
go: finding github.com/stretchr/testify v1.2.2
go: finding github.com/davecgh/go-spew v1.1.1
go: finding github.com/DefinitelyMod/gocsv v0.0.0-20181205141819-acfa5f112b45
go: finding github.com/pmezard/go-difflib v1.0.0
[novalagung:chapter-A.60-go-modules $ tree -L 1
.
├── go.mod
├── go.sum
└── main.go
└── models
└── vendor

2 directories, 3 files
[novalagung:chapter-A.60-go-modules $ go run main.go
{u001 Noval}
17
novalagung:chapter-A.60-go-modules $ ] needed to build and test all
```

Jika pada sebuah projek sudah enabled `dep`, dan ingin di enable Go Modules. Maka file `Gopkg.lock` yang sudah ada akan dikonversi ke-dalam bentuk `go.mod` dan `go.sum`.

## A.60.5. Sinkronisasi Dependencies

Gunakan command `go mod tidy` untuk sinkronisasi dependencies yang digunakan dalam project. Dengan command tersebut, secara otomatis 3rd party yang belum ditambahkan akan dicatat dan ditambahkan; dan yang tidak digunakan tapi terlanjur tercatat akan dihapuskan.

## B.1. Golang Web App: Hello World

Pada bab pertama ini, kita akan belajar bagaimana cara membuat aplikasi web "Hello World" sederhana menggunakan Golang.

### B.1.1. Pembuatan Aplikasi

Pertama buat folder projek baru dengan isi `main.go`, tentukan package-nya sebagai `main`, lalu import package `fmt` dan `net/http`.

```
package main

import "fmt"
import "net/http"
```

Setelah itu, siapkan dua buah fungsi, masing-masing fungsi memiliki skema parameter yang sama seperti berikut.

- Parameter ke-1 bertipe `http.ResponseWriter`
- Parameter ke-2 bertipe `*http.Request`

Fungsi dengan struktur di atas diperlukan oleh `http.HandleFunc` untuk keperluan penanganan request ke rute yang ditentukan.

Berikut merupakan dua fungsi yang dimaksud.

```
func handlerIndex(w http.ResponseWriter, r *http.Request) {
    var message = "Welcome"
    w.Write([]byte(message))
}

func handlerHello(w http.ResponseWriter, r *http.Request) {
    var message = "Hello world!"
    w.Write([]byte(message))
}
```

Method `Write()` milik parameter pertama (yang bertipe `http.ResponseWriter`), digunakan untuk meng-output-kan nilai balik data. Argumen method adalah data yang ingin dijadikan output, ditulis dalam bentuk `[]byte`.

Pada contoh ini, data yang akan kita tampilkan bertipe string, maka perlu dilakukan casting dari `string` ke `[]byte`. Contohnya bisa dilihat seperti pada kode di atas, di bagian `w.Write([]byte(message))`.

Selanjutnya, siapkan fungsi `main()` dengan isi di dalamnya adalah beberapa rute atau route, dengan aksi adalah kedua fungsi yang sudah disiapkan di atas. Tak lupa siapkan juga kode untuk start server.

```
func main() {
    http.HandleFunc("/", handlerIndex)
    http.HandleFunc("/index", handlerIndex)
    http.HandleFunc("/hello", handlerHello)

    var address = "localhost:9000"
    fmt.Printf("server started at %s\n", address)
    err := http.ListenAndServe(address, nil)
    if err != nil {
        fmt.Println(err.Error())
    }
}
```

Fungsi `http.HandleFunc()` digunakan untuk routing. Parameter pertama adalah rute dan parameter ke-2 adalah handler-nya.

Fungsi `http.ListenAndServe()` digunakan membuat sekaligus start server baru, dengan parameter pertama adalah alamat web server yang diinginkan (bisa diisi host, host & port, atau port saja). Parameter kedua merupakan object mux atau multiplexer.

Dalam chapter ini kita menggunakan default mux yang sudah disediakan oleh golang, jadi untuk parameter ke-2 cukup isi dengan `nil`.

## B.1.2. Testing

Program sudah siap, jalankan menggunakan `go run`.

```
[novalagung:chapter-1.1 $ go run main.go
server started at localhost:9000]
```

Cek pada browser rute yang sudah dibuat, output akan muncul.



Berikut merupakan penjelasan detail per-bagian program yang telah kita buat dari contoh di atas.

## B.1.3. Penggunaan `http.HandleFunc()`

Fungsi ini digunakan untuk **routing**, menentukan aksi dari sebuah url tertentu ketika diakses (di sini url tersebut kita sebut sebagai rute/route). Rute dituliskan dalam `string` sebagai parameter pertama, dan aksi-nya sendiri dibungkus dalam fungsi (bisa berupa closure) yang ditempatkan pada parameter kedua (kita sebut sebagai handler).

Pada kode di atas, tiga buah rute didaftarkan:

- Rute `/` dengan aksi adalah fungsi `handlerIndex()`
- Rute `/index` dengan aksi adalah sama dengan `/`, yaitu fungsi `handlerIndex()`
- Rute `/hello` dengan aksi fungsi `handlerHello()`

Ketika rute-rute tersebut diakses lewat browser, outpunnnya adalah isi-handler dari rute yang bersangkutan. Kebetulan pada bab ini, ketiga rute tersebut outputnya adalah sama, yaitu berupa string.

Pada contoh di atas, ketika rute yang tidak terdaftar diakses, secara otomatis handler rute `/` akan terpanggil.

## B.1.4. Penjelasan Mengenai Handler

Route handler atau handler atau parameter kedua fungsi `http.HandleFunc()`, adalah sebuah fungsi dengan ber-skema `func (ResponseWriter, *Request)`.

- Parameter ke-1 merupakan objek untuk keperluan http response.
- Sedang parameter ke-2 yang bertipe `*request` ini, berisikan informasi-informasi yang berhubungan dengan http request untuk rute yang bersangkutan.

Contoh penulisan handler bisa dilihat pada fungsi `handlerIndex()` berikut.

```
func handlerIndex(w http.ResponseWriter, r *http.Request) {
    var message = "Welcome"
```

```
w.Write([]byte(message))
}
```

Output dari rute, dituliskan di dalam handler menggunakan method `write()` milik objek `ResponseWriter` (parameter pertama). Output bisa berupa apapun, untuk output text tinggal lakukan casting dari tipe `string` ke `[]byte`, aturan ini juga berlaku untuk beberapa jenis output lainnya seperti html dan json, namun response header `Content-Type` perlu disesuaikan.

Pada contoh program yang telah kita buat, handler `Index()` memunculkan text `"Welcome"`, dan handler `Hello()` memunculkan text `"Hello world!"`.

Sebuah handler bisa dipergunakan pada banyak rute, bisa dilihat pada di atas handler `Index()` digunakan pada rute `/` dan `/index`.

## B.1.5. Penggunaan `http.ListenAndServe()`

Fungsi ini digunakan untuk membuat web server baru. Pada contoh yang telah dibuat, web server di-start pada port `9000` (bisa dituliskan dalam bentuk `localhost:9000` atau cukup `:9000` saja).

```
var address = ":9000"
fmt.Printf("server started at %s\n", address)
err := http.ListenAndServe(address, nil)
```

Fungsi `http.ListenAndServe()` bersifat blocking, menjadikan semua statement setelahnya tidak akan dieksekusi, sebelum di-stop.

Fungsi ini mengembalikan nilai balik ber-tipe `error`. Jika proses pembuatan web server baru gagal, maka kita bisa mengetahui root-cause nya apa.

## B.1.6. Web Server Menggunakan `http.Server`

Selain menggunakan `http.ListenAndServe()`, ada cara lain yang bisa diterapkan untuk start web server, yaitu dengan memanfaatkan struct `http.Server`.

Kode di bagian start server yang sudah kita buat, jika diubah ke cara ini, kurang lebih menjadi seperti berikut.

```
var address = ":9000"
fmt.Printf("server started at %s\n", address)

server := new(http.Server)
server.Addr = address
err := server.ListenAndServe()
if err != nil {
    fmt.Println(err.Error())
}
```

Informasi host/port perlu dimasukan dalam property `.Addr` milik objek server. Lalu dari objek tersebut panggil method `.ListenAndServe()` untuk start web server.

Kelebihan menggunakan `http.Server` salah satunya adalah kemampuan untuk mengubah beberapa konfigurasi default web server golang.

Contoh, pada kode berikut, timeout untuk read request dan write request di ubah menjadi 10 detik.

```
server.ReadTimeout = time.Second * 10
```

```
server.WriteTimeout = time.Second * 10
```

Ada banyak lagi property dari struct `http.Server` ini, yang pastinya akan dibahas pada bab-bab selanjutnya.

## B.2. Routing http.HandleFunc

Dalam Golang, routing bisa dilakukan dengan beberapa cara, diantaranya:

1. Dengan memanfaatkan fungsi `http.HandleFunc()`
2. Mengimplementasikan interface `http.Handler` pada suatu struct, untuk kemudian digunakan pada fungsi `http.Handle()`
3. Membuat multiplexer sendiri dengan memanfaatkan struct `http.ServeMux`

Di buku ini ketiga cara tersebut akan dibahas. Namun khusus pada bab ini saja, hanya `http.HandleFunc()` yang kita pelajari.

Metode routing cara pertama dan cara kedua memiliki kesamaan yaitu sama-sama menggunakan `DefaultServeMux` untuk pencocokan rute yang diregistrasikan. Mengenai apa itu `DefaultServeMux` akan kita bahas lebih mendetail pada bab lain.

### B.2.1. Penggunaan http.HandleFunc()

Seperti yang sudah dijelaskan sekilas pada bab sebelumnya, fungsi `http.HandleFunc()` digunakan untuk registrasi rute dan handler-nya.

Penggunaan fungsi ini cukup mudah, panggil saja fungsi lalu isi dua parameternya.

1. Parameter ke-1, adalah rute. Sebagai contoh: `/`, `/index`, `/about`.
2. Parameter ke-2, berisikan handler untuk rute bersangkutan. Sebagai contoh handler untuk rute `/` bertugas untuk menampilkan output berupa html `<p>hello</p>`.

Agar lebih mudah dipahami mari kita praktikan. Siapkan sebuah file `main.go` dengan package adalah `main`, dan import package `net/http` didalamnya.

```
package main

import "fmt"
import "net/http"
```

Buat fungsi `main()`, didalamnya siapkan sebuah closure `handlerIndex`, lalu gunakan closure tersebut sebagai handler dari dua rute baru yang diregistrasi, yaitu `/` dan `/index`.

```
func main() {
    handlerIndex := func(w http.ResponseWriter, r *http.Request) {
        w.Write([]byte("hello"))
    }

    http.HandleFunc("/", handlerIndex)
    http.HandleFunc("/index", handlerIndex)
}
```

Selanjutnya, masih dalam fungsi `main()`, tambahkan rute baru `/data` dengan handler adalah anonymous function.

```
func main() {
    // ...

    http.HandleFunc("/data", func(w http.ResponseWriter, r *http.Request) {
        w.Write([]byte("hello again"))
    })
}
```

```
}
```

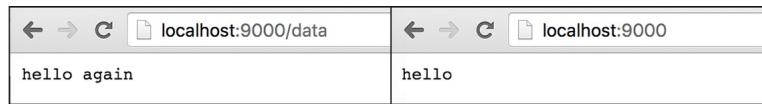
Terakhir, jalankan server.

```
func main() {
    // ...

    fmt.Println("server started at localhost:9000")
    http.ListenAndServe(":9000", nil)
}
```

## B.2.2. Test

Tes dan lihat hasilnya.



Dalam routing, handler bisa berupa fungsi, closure, ataupun anonymous function. Yang terpenting adalah skema fungsi-nya sesuai dengan `func (http.ResponseWriter, *http.Request)`.

## B.3. Routing Static Assets

Pada bagian ini kita akan belajar bagaimana cara routing static assets atau static contents. Seperti file css, js, gambar, umumnya dikategorikan sebagai static assets.

### B.3.1. Struktur Aplikasi

Buat project baru, siapkan file dan folder dengan struktur sesuai dengan gambar berikut.

Name	Date Modified
chapter-1.4	Today, 8:25 PM
assets	Today, 8:25 PM
site.css	Today, 8:25 PM
main.go	Jan 15, 2016, 1:40 PM

Dalam folder `assets`, isi dengan file apapun, bisa gambar atau file js. Selanjutnya masuk ke bagian routing static assets.

### B.3.2. Routing

Berbeda dengan routing menggunakan `http.HandleFunc()`, routing static assets lebih mudah. Silakan tulis kode berikut dalam `main.go`, setelahnya kita akan bahas secara mendetail.

```
package main

import "fmt"
import "net/http"

func main() {
    http.Handle("/static/",
        http.StripPrefix("/static/",
            http.FileServer(http.Dir("assets"))))

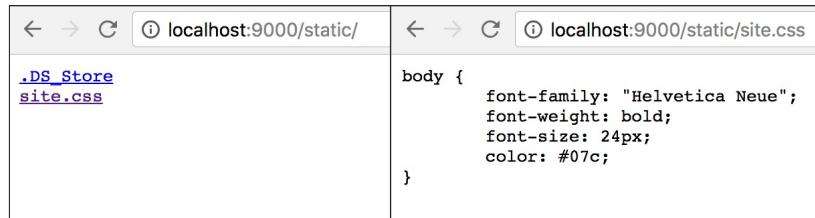
    fmt.Println("server started at localhost:9000")
    http.ListenAndServe(":9000", nil)
}
```

Syarat yang dibutuhkan untuk routing static assets masih sama dengan routing handler, yaitu perlu didefiniskan rute-nya dan handler-nya. Hanya saja pembedanya, dalam routing static assets yang digunakan adalah `http.Handle()`, bukan `http.HandleFunc()`.

- Rute terpilih adalah `/static/`, maka nantinya semua request yang di-awali dengan `/static/` akan diarahkan ke sini. Registrasi rute menggunakan `http.Handle()` adalah berbeda dengan routing menggunakan `http.HandleFunc()`, lebih jelasnya akan ada sedikit penjelasan di bab lain.
- Sedang untuk handler-nya bisa di-lihat, ada pada parameter ke-2 yang isinya statement `http.StripPrefix()`. Sebenarnya actual handler nya berada pada `http.FileServer()`. Fungsi `http.StripPrefix()` hanya digunakan untuk membungkus actual handler.

Fungsi `http.FileServer()` mengembalikan objek ber-tipe `http.Handler`. Fungsi ini berguna untuk men-serve semua http request, dengan konten yang didefinisikan pada parameter. Pada konteks ini yang di-maksud adalah `http.Dir("assets")`. Semua konten, entah file ataupun folder, yang ada di dalam folder `assets` akan di proses dalam handler.

Jalankan `main.go`, lalu test hasilnya di browser `http://localhost:9000/static/`.



### B.3.3. Penjelasan

Penjelasan akan lebih mudah dipahami jika disajikan juga contoh praktik, maka sejenak kita coba bahas menggunakan contoh sederhana berikut.

```
http.Handle("/", http.FileServer(http.Dir("assets")))
```

Jika dilihat pada struktur folder yang sudah di-buat, di dalam folder `assets` terdapat file bernama `site.css`. Maka dengan bentuk routing pada contoh sederhana di atas, request ke `/site.css` akan diarahkan ke path `./site.css` (relatif dari folder `assets`). Permisalan contoh lainnya:

- Request ke `/site.css` mengarah path `./site.css` relatif dari folder `assets`
- Request ke `/script.js` mengarah path `./script.js` relatif dari folder `assets`
- Request ke `/some/folder/test.png` mengarah path `./some/folder/test.png` relatif dari folder `assets`
- ... dan seterusnya

Fungsi `http.Dir()` berguna untuk adjustment path parameter. Separator dari path yang di-definisikan akan otomatis di-konversi ke path separator sesuai sistem operasi.

Contoh selanjutnya, silakan perhatikan kode berikut.

```
http.Handle("/static", http.FileServer(http.Dir("assets")))
```

Hasil dari routing:

- Request ke `/static/site.css` mengarah ke `./static/site.css` relatif dari folder `assets`
- Request ke `/static/script.js` mengarah ke `./static/script.js` relatif dari folder `assets`
- Request ke `/static/some/folder/test.png` mengarah ke `./static/some/folder/test.png` relatif dari folder `assets`
- ... dan seterusnya

Terlihat bahwa rute yang didaftarkan juga akan digabung dengan path destinasi file yang dicari, dan ini menjadikan path tidak valid. File `site.css` berada pada path `assets/site.css`, sedangkan dari routing di atas pencarian file mengarah ke path `assets/static/site.css`. Di sinilah kegunaan dari fungsi `http.StripPrefix()`.

Fungsi `http.StripPrefix()` ini berguna untuk menghapus prefix dari endpoint yang di-request. Pada contoh paling atas, request ke url yang di-awali dengan `/static/` hanya akan di ambil url setelahnya.

- Request ke `/static/site.css` menjadi `/site.css`
- Request ke `/static/script.js` menjadi `/script.js`
- Request ke `/static/some/folder/test.png` menjadi `/some/folder/test.png`
- ... dan seterusnya

Routing static assets menjadi valid, karena file yang di-request akan cocok dengan path folder dari file yang di request.



## B.4. Template: Render HTML Template

Pada bagian ini kita akan belajar bagaimana cara render file **template** ber-tipe **HTML**, untuk ditampilkan pada browser.

Terdapat banyak jenis template pada golang, yang akan kita pakai adalah template HTML. Package `html/template` menyediakan banyak sekali fungsi untuk kebutuhan rendering dan parsing file template jenis ini.

### B.4.1. Struktur Aplikasi

Buat project baru, siapkan file dan folder dengan struktur sesuai dengan gambar berikut.

Name	Date Modified
chapter-1.4	Today, 8:25 PM
assets	Today, 8:25 PM
site.css	Today, 8:25 PM
main.go	Jan 15, 2016, 1:40 PM
views	Jan 15, 2016, 10:34 AM
index.html	Today, 2:54 PM

### B.4.2. Back End

Hal pertama yang perlu dilakukan adalah mempersiapkan back end. Buka file `main.go`, import package `net/http`, `html/template`, dan `path`. Siapkan juga rute `/`.

```
package main

import "fmt"
import "net/http"
import "html/template"
import "path"

func main() {
    http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
        // not yet implemented
    })
    fmt.Println("server started at localhost:9000")
    http.ListenAndServe(":9000", nil)
}
```

Handler rute `/` akan kita isi dengan proses untuk rendering template html untuk ditampilkan ke layar browser. Beberapa data disisipkan dalam proses rendering template.

Silakan tulis kode berikut di dalam handler rute `/`.

```
var filepath = path.Join("views", "index.html")
var tmpl, err = template.ParseFiles(filepath)
if err != nil {
    http.Error(w, err.Error(), http.StatusInternalServerError)
    return
}

var data = map[string]interface{}{
    "title": "Learning Golang Web",
    "name": "Batman",
```

```

    }

err = tmpl.Execute(w, data)
if err != nil {
    http.Error(w, err.Error(), http.StatusInternalServerError)
}

```

Package `path` berisikan banyak fungsi yang berhubungan dengan lokasi folder atau path, yang salah satu diantaranya adalah fungsi `path.Join()`. Fungsi ini digunakan untuk menggabungkan folder atau file atau keduanya menjadi sebuah path, dengan separator relatif terhadap OS yang digunakan.

Separator yang digunakan oleh `path.Join()` adalah `\` untuk windows dan `/` untuk unix.

Contoh penerapan `path.Join()` bisa dilihat di kode di atas, `views` di-join dengan `index.html`, menghasilkan `views/index.html`.

Sedangkan `template.ParseFiles()`, digunakan untuk parsing file template, dalam contoh ini file `view/index.html`. Fungsi ini mengembalikan 2 data, yaitu hasil dari proses parsing yang bertipe `*template.Template`, dan informasi `error` jika ada.

Fungsi `http.Error()` digunakan untuk menandai response (`http.ResponseWriter`) bahwa terjadi error, dengan kode error dan pesan error bisa ditentukan. Pada contoh di atas yang digunakan adalah **500 - internal server error** yang direpresentasikan oleh variabel `http.StatusInternalServerError`.

Method `Execute()` milik `*template.Template`, digunakan untuk menyisipkan data pada template, untuk kemudian ditampilkan ke browser. Data bisa disisipkan dalam bentuk `struct`, `map`, atau `interface{}`.

- Jika dituliskan dalam bentuk `map`, maka **key** akan menjadi nama variabel dan **value** menjadi nilainya
- Jika dituliskan dalam bentuk variabel objek cetakan `struct`, nama **property** akan menjadi nama variabel

Pada contoh di atas, data map yang berisikan key `title` dan `name` disisipkan ke dalam template yang sudah di parsing.

### B.4.3. Front End

OK, back end sudah siap, selanjutnya kita masuk ke bagian user interface. Pada file `views/index.html`, tuliskan kode html sederhana berikut.

```

<!DOCTYPE html>
<html>
    <head>
        <title>{{.title}}</title>
    </head>
    <body>
        <p>Welcome {{.name}}</p>
    </body>
</html>

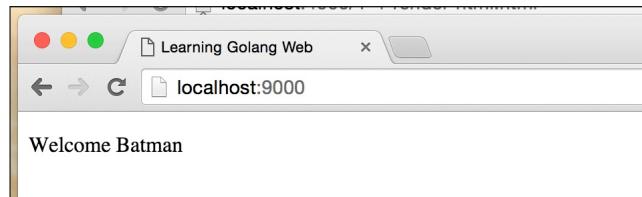
```

Untuk menampilkan variabel yang disisipkan ke dalam template, gunakan notasi `{{.namaVariabel}}`. Pada contoh di atas, data `title` dan `name` yang dikirim dari back end ditampilkan.

Tanda titik `.` pada `{{.namaVariabel}}` menerangkan bahwa variabel tersebut diakses dari **current scope**. Dan current scope default adalah data `map` atau objek yang dilempar back end.

### B.4.4. Testing

Semua sudah siap, maka jalankan program lalu lakukan testing via browser.



## B.4.5. Static File CSS

Kita akan coba tambahkan sebuah stylesheet disini. Langsung saja, buat file statis `assets/site.css`, isi dengan kode berikut.

```
body {
    font-family: "Helvetica Neue";
    font-weight: bold;
    font-size: 24px;
    color: #07c;
}
```

Pada `views/index.html`, include-kan file css.

```
<link rel="stylesheet" href="/static/site.css" />
```

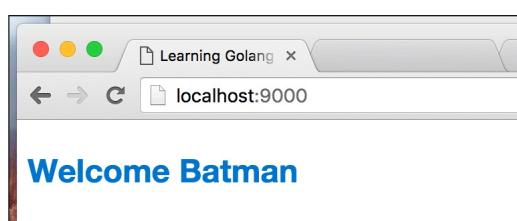
Terakhir pada fungsi `main()`, tambahkan router untuk handling file statis.

```
func main() {
    // ...

    http.Handle("/static/",
        http.StripPrefix("/static/",
            http.FileServer(http.Dir("assets"))))

    fmt.Println("server started at localhost:9000")
    http.ListenAndServe(":9000", nil)
}
```

Jalankan aplikasi untuk test hasil.



## B.5. Template: Render Partial HTML Template

Satu buah halaman yang berisikan html, bisa terbentuk dari banyak template html (partial). Pada bab ini kita akan belajar bagaimana membuat, mem-parsing, dan me-render semua file tersebut.

Ada beberapa metode yang bisa digunakan, dari kesemuanya akan kita bahas 2 diantaranya, yaitu:

- Menggunakan fungsi `template.ParseGlob()` .
- Menggunakan fungsi `template.ParseFiles()` .

### B.5.1. Struktur Aplikasi

Mari langsung kita praktikan. Buat project baru, siapkan file dan folder dengan susunan seperti dengan gambar berikut.

Name	Date Modified
chapter-1.5	Today, 6:33 PM
main.go	Today, 6:33 PM
views	Today, 6:33 PM
_header.html	Today, 6:32 PM
_message.html	Today, 6:32 PM
about.html	Today, 6:32 PM
index.html	Today, 6:19 PM

### B.5.2. Back End

Buka `main.go` , isi dengan kode berikut.

```
package main

import (
    "net/http"
    "html/template"
    "fmt"
)

type M map[string]interface{}

func main() {
    var tmpl, err = template.ParseGlob("views/*")
    if err != nil {
        panic(err.Error())
        return
    }
}
```

Tipe `M` merupakan alias dari `map[string]interface{}` , disiapkan untuk mempersingkat penulisan tipe map tersebut. Di bab-bab selanjutnya kita akan banyak menggunakan tipe ini.

Pada kode di atas, di dalam fungsi `main()` , fungsi `template.ParseGlob()` dipanggil, dengan parameter adalah pattern path `"views/*"` . Fungsi ini digunakan untuk memparsing semua file yang match dengan pattern yang ditentukan, dan fungsi ini mengembalikan 2 objek: `*template.Template & error` .

Pattern path pada fungsi `template.ParseGlob()` nantinya akan diproses oleh `filepath.Glob()`

Proses parsing semua file html dalam folder `views` dilakukan di-awal, agar ketika mengakses rute tertentu yang menampilkan html, tidak terjadi proses parsing lagi.

Parsing semua file menggunakan `template.ParseGlob()` yang dilakukan di luar handler, tidak direkomendasikan dalam fase development. Karena akan mempersulit testing html. Lebih detailnya akan dibahas di bagian bawah.

Selanjutnya, masih di dalam fungsi `main()`, siapkan 2 buah rute.

```
http.HandleFunc("/index", func(w http.ResponseWriter, r *http.Request) {
    var data = M{"name": "Batman"}
    err = tmpl.ExecuteTemplate(w, "index", data)
    if err != nil {
        http.Error(w, err.Error(), http.StatusInternalServerError)
    }
})

http.HandleFunc("/about", func(w http.ResponseWriter, r *http.Request) {
    var data = M{"name": "Batman"}
    err = tmpl.ExecuteTemplate(w, "about", data)
    if err != nil {
        http.Error(w, err.Error(), http.StatusInternalServerError)
    }
})

fmt.Println("server started at localhost:9000")
http.ListenAndServe(":9000", nil)
```

Kedua rute tersebut sama, pemembedanya adalah template yang di-render. Rute `/index` me-render template bernama `index`, dan rute `/about` me-render template bernama `about`.

Karena semua file html sudah diparsing di awal, maka untuk render template tertentu cukup dengan memanggil method `ExecuteTemplate()`, dengan menyisipkan 3 parameter berikut:

1. Parameter ke-1, objek `http.ResponseWriter`
2. Parameter ke-2, nama template
3. Parameter ke-3, data

Nama template bukanlah nama file. Setelah masuk ke bagian front-end, akan diketahui apa yang dimaksud dengan nama template.

### B.5.3. Front End

- **Template `index.html`**

OK, sekarang waktunya untuk mulai menyiapkan template view. Ada 4 buah template yang harus kita siapkan satu per satu.

Buka file `index.html`, lalu tulis kode berikut.

```
{{define "index"}}
<!DOCTYPE html>
<html>
    <head>
        {{template "_header"}}
    </head>
    <body>
        {{template "_message"}}
        <p>Page: Index</p>
        <p>Welcome {{.name}}</p>
```

```
</body>
</html>
{{end}}
```

Pada kode di atas terlihat bahwa ada beberapa kode yang ditulis dengan notasinya `{{ }}`. Berikut adalah penjelasannya.

- Statement  `{{define "index"}}` , digunakan untuk mendefinisikan nama template. Semua blok kode setelah statement tersebut (batasnya adalah hingga statement  `{{end}}` ) adalah milik template dengan nama  `index` . keyword  `define`  digunakan dalam penentuan nama template.
- Statement  `{{template "_header"}}`  artinya adalah template bernama  `_header`  di-include ke bagian itu. keyword  `template`  digunakan untuk include template lain.
- Statement  `{{template "_message"}}` , sama seperti sebelumnya, template bernama  `_message`  akan di-include.
- Statement  `{{.name}}`  akan memunculkan data,  `name` , yang data ini sudah disisipkan oleh back end pada saat rendering.
- Statement  `{{end}}`  adalah penanda batas akhir pendefinisian template.

### • **Template `about.html`**

Template ke-2,  `about.html`  diisi dengan dengan kode yang sama seperti pada  `index.html` , hanya berbeda di bagian nama template dan beberapa text.

```
 {{define "about"}}
<!DOCTYPE html>
<html>
  <head>
    {{template "_header"}}
  </head>
  <body>
    {{template "_message"}}
    <p>Page: About</p>
    <p>Welcome {{.name}}</p>
  </body>
</html>
{{end}}
```

### • **Template `_header.html`**

Buka file  `_header.html` , definisikan template bernama  `_header`  dengan isi adalah judul halaman.

```
 {{define "_header"}}
<title>Learn Golang Template</title>
{{end}}
```

Nama file bisa ditulis dengan diawali karakter underscore atau  `_` . Pada bab ini, nama file yang diawali  `_`  kita asumsikan sebagai template parsial, template yang nantinya di-include-kan ke template utama.

### • **Template `_message.html`**

Definisikan juga template  `_message`  pada file  `_message.html` . Isinya sebuah pesan.

```
 {{define "_message"}}
<p>Welcome</p>
{{end}}
```

## B.5.5. Test

Jalankan aplikasi, test via browser.



Bisa dilihat pada gambar di atas, ketika rute `/index` dan `/about` di akses, konten yang keluar adalah berbeda, sesuai dengan template yang di-render di masing-masing rute.

## B.5.6. Parsing Banyak File HTML Menggunakan `template.ParseFiles()`

Metode parsing menggunakan `template.ParseGlob()` memiliki kekurangan yaitu sangat tergantung terhadap pattern path yang digunakan. Jika dalam suatu proyek terdapat sangat banyak file html dan folder, sedangkan hanya beberapa yang digunakan, pemilihan pattern path yang kurang tepat akan menjadikan file lain ikut ter-parsing dengan sia-sia.

Dan juga, karena statement `template.ParseGlob()` dieksekusi diluar handler, maka ketika ada perubahan pada salah satu view, lalu halaman di refresh, output yang dihasilkan akan tetap sama. Solusi dari masalah ini adalah dengan memanggil `template.ParseGlob()` di tiap handler rute-rute yang diregistrasikan.

Best practices yang bisa diterapkan, ketika environment adalah production, maka tempatkan `template.ParseGlob()` di luar (sebelum) handler. Sedangkan pada environment development, taruh `template.ParseGlob()` di dalam masing-masing handler. Gunakan seleksi kondisi untuk mengakomodir skenario ini.

Alternatif metode lain yang bisa digunakan, yang lebih efisien, adalah dengan memanfaatkan fungsi `template.ParseFiles()`. Fungsi ini selain bisa digunakan untuk parsing satu buah file saja (seperti yang sudah dicontohkan di bab sebelumnya), bisa digunakan untuk parsing banyak file.

Mari kita praktikan. Ubah handler rute `/index` dan `/about`. Gunakan `template.ParseFiles()` dengan isi parameter (variadic) adalah path dari file-file html yang akan dipergunakan di masing-masing rute. Lalu hapus statement `template.ParseGlob()`

- Rute `/index` dan handlernya.

```
http.HandleFunc("/index", func(w http.ResponseWriter, r *http.Request) {
    var data = M{"name": "Batman"}
    var tmpl = template.Must(template.ParseFiles(
        "views/index.html",
        "views/_header.html",
        "views/_message.html",
    ))
    var err = tmpl.ExecuteTemplate(w, "index", data)
    if err != nil {
        http.Error(w, err.Error(), http.StatusInternalServerError)
    }
})
```

- Rute `/about` dan handlernya.

```

http.HandleFunc("/about", func(w http.ResponseWriter, r *http.Request) {
    var data = M{"name": "Batman"}
    var tmpl = template.Must(template.ParseFiles(
        "views/about.html",
        "views/_header.html",
        "views/_message.html",
    ))
    var err = tmpl.ExecuteTemplate(w, "about", data)
    if err != nil {
        http.Error(w, err.Error(), http.StatusInternalServerError)
    }
})

```

- **Hapus statement** `template.ParseGlob()` .

```

var tmpl, err = template.ParseGlob("views/*")
if err != nil {
    panic(err.Error())
    return
}

```

Rute `/index` memakai view `_header.html` , `_message.html` , dan `index.html` ; sedangkan rute `/about` tidak memakai `index.html` , melainkan `about.html` .

Wrap fungsi `template.ParseFiles()` dalam `template.Must()` . Fungsi ini berguna untuk deteksi error pada saat membuat instance `*template.Template` baru atau ketika sedang mengolahnya. Ketika ada error, `panic` dimunculkan.

Jalankan aplikasi untuk mengetes hasilnya.

## B.6. Template: Actions & Variables

**Actions** adalah *predefined* keyword yang sudah disiapkan oleh golang, yang biasa dimanfaatkan dalam pembuatan template.

Sebenarnya pada dua bab sebelumnya, secara tidak sadar kita telah menggunakan beberapa jenis actions, diantaranya:

- Penggunaan **pipeline output**. Nilai yang diapit tanda `{}{ }{ }`, yang nantinya akan dimunculkan di layar sebagai output, contohnya: `{}{"hello world"}{ }`.
- Include template lain menggunakan keyword `template`, contohnya: `{}{{template "name"}}{ }`.

Pada bab ini, kita akan belajar lebih banyak lagi tentang actions lain yang disediakan golang, juga cara pembuatan dan pemanfaatan variabel pada template view.

### B.6.1. Persiapan

Pertama-tama, siapkan sebuah file bernama `main.go`, lalu isi dengan kode berikut.

```
package main

import "net/http"
import "fmt"
import "html/template"

type Info struct {
    Affiliation string
    Address     string
}

type Person struct {
    Name      string
    Gender    string
    Hobbies   []string
    Info      Info
}
```

Pada kode di atas, dua buah struct disiapkan, `Info` dan `Person` (yang dimana struct `Info` di-embed ke dalam struct `Person`). Kedua struct tersebut nantinya akan digunakan untuk mencetak objek, yang kemudian object tersebut disisipkan kedalam view.

Selanjutnya, siapkan fungsi `main()`, dengan didalamnya berisikan 1 buah route handler `/`, dan juga kode untuk menjalankan server pada port `9000`.

```
func main() {
    http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
        var person = Person{
            Name:      "Bruce Wayne",
            Gender:    "male",
            Hobbies:   []string{"Reading Books", "Traveling", "Buying things"},
            Info:      Info{"Wayne Enterprises", "Gotham City"},
        }

        var tmpl = template.Must(template.ParseFiles("view.html"))
        if err := tmpl.Execute(w, person); err != nil {
            http.Error(w, err.Error(), http.StatusInternalServerError)
        }
    })
}
```

```

    fmt.Println("server started at localhost:9000")
    http.ListenAndServe(":9000", nil)
}

```

Pada route handler `/` di atas, variabel objek `person` dibuat, lalu disisipkan sebagai data pada view `view.html` yang sebelumnya sudah diparsing.

Perlu diketahui, ketika data yang disisipkan ke view berbentuk `map`, maka `key` (yang nantinya akan menjadi nama variabel) boleh dituliskan dalam huruf kecil. Sedangkan jika berupa variabel objek cetakan `struct`, property harus dituliskan public (huruf pertama kapital).

Data yang disisipkan ke view, jika tipe nya adalah struct, maka hanya properties ber-modifier public (ditandai dengan huruf kapital di awal nama property) yang bisa diakses dari view.

OK, bagian back end sudah selesai, sekarang saatnya lanjut ke bagian depan. Buat file view baru bernama `view.html`, isi dengan kode berikut.

```

<html>
  <head>
    <title>Learning html/template Actions</title>
  </head>
  <body>
    <table>
    </table>
  </body>
</html>

```

Selanjutnya silakan ikuti step-step berikut.

## B.6.2. Pipeline Output & Komentar

Actions pertama yang akan kita coba terapkan adalah pipeline output, menampilkan output ke layar. Caranya cukup mudah, cukup dengan menuliskan apa yang ingin ditampilkan di layar dengan diapit tanda `{{ }}` (bisa berupa variabel yang dilempar dari back end, bisa juga literal string).

Tulis kode berikut di dalam tag `<table></table>` pada `view.html`.

```

<tr>
  {{/* example how to use actions */}}
  <td>{{"Name"}}</td>
  <td>: {{.Name}}</td>
</tr>

```

Test hasilnya pada browser.



Untuk menampilkan tipe data lain selain string, caranya masih sama, langsung dituliskan dalam `{{ }}`. Untuk menampilkan nilai variabel, caranya juga masih sama, hanya saja perlu ditambahkan tanda titik `.` pada penulisannya (tanda titik `.` adalah penanda bahwa variabel tersebut adalah variabel terluar; bukan merupakan elemen array, item map, atau property struct).

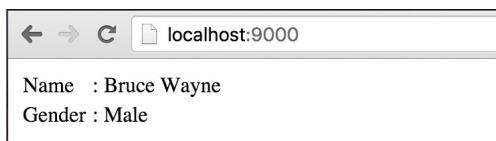
Sedangkan untuk komentar, gunakan tanda `{{/* */}}`. Komentar tidak akan dimunculkan dalam output.

### B.6.3. Membuat & Menampilkan Isi Variabel

Cara membuat variabel dalam template adalah dengan mendeklarasikannya menggunakan operator `:=`, dengan ketentuan nama variabel harus diawali dengan tanda dollar `$`.

```
<tr>
    <td>Gender</td>
    {{$gender := .Gender}}
    <td style="text-transform: capitalize;">
        {{$gender}}
    </td>
</tr>
```

Jika ingin menampilkan isi variabel, tuliskan sebagai pipeline.

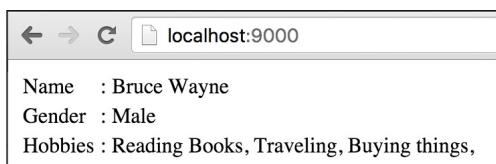


### B.6.4. Perulangan

Actions `range` digunakan untuk melakukan perulangan pada template view. Keyword ini bisa diterapkan pada tipe data `map` atau array. Cara penggunaannya sedikit berbeda dibanding penggunaan range pada Golang. Silakan perhatikan contoh berikut.

```
<tr>
    <td>Hobbies</td>
    <td>:
        {{range $index, $elem := .Hobbies}}
            {{$elem}},
        {{end}}
    </td>
</tr>
```

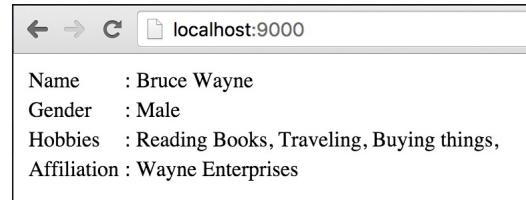
Penulisannya cukup unik, keyword `range` dituliskan terlebih dahulu, diikuti variabel penampung index dan elemen. Jika yang dibutuhkan hanya elemen saja, bisa cukup gunakan `{{range $elem := .Hobbies}}`. Semua kode setelah baris deklarasi hingga penutup  `{{end}}`, akan diulang sesuai jumlah elemen/item-nya.



### B.6.5. Pengaksesan Property Variabel Objek

Cara mengakses property sebuah variabel objek bisa dilakukan lewat notasi titik `.`, dengan ketentuan property tersebut bermodifier public.

```
<tr>
    <td>Affiliation</td>
    <td>: {{$Info.Affiliation}}</td>
</tr>
```

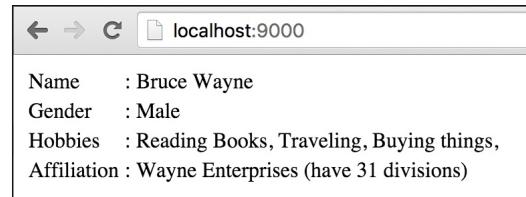


Sedangkan untuk pengaksesan method, caranya juga sama, hanya saja tidak perlu dituliskan tanda kurung method-nya. Buat sebuah method pada struct `Info`.

```
func (t Info) GetAffiliationDetailInfo() string {
    return "have 31 divisions"
}
```

Lalu akses method tersebut pada template view.

```
<tr>
    <td>Affiliation</td>
    <td>: {{.Info.Affiliation}} ({{.Info.GetAffiliationDetailInfo}})</td>
</tr>
```



Lalu bagaimana cara pengaksesan method yang membutuhkan parameter, jika tanda kurungnya tidak boleh dituliskan? Jawabannya akan kita temukan pada bab selanjutnya.

## B.6.6. Penggunaan Keyword `with` Untuk Mengganti Scope Variabel Pada Suatu Blok

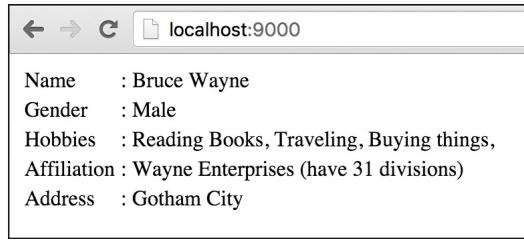
Secara default `current scope` di template view adalah data yang dilempar back end. Scope current objek bisa diganti dengan menggunakan keyword `with`, sehingga nantinya untuk mengakses sub-property variabel objek (seperti `.Info.Affiliation`), bisa tidak dilakukan dari objek terluar.

Current scope yg dimaksud di sini adalah seperti object `this` ibarat bahasa pemrograman lain.

Sebagai contoh property `Info` yang merupakan variabel objek. Kita bisa menentukan scope suatu block adalah mengikuti variabel objek tersebut.

```
{{with .Info}}
<tr>
    <td>Address</td>
    <td>: {{.Address}}</td>
</tr>
{{end}}
```

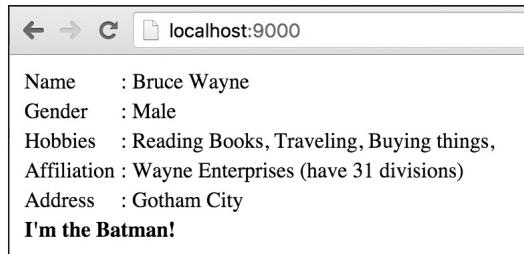
Pada contoh di atas, sebuah blok ditentukan scope-nya adalah `Info`. Maka di dalam blok kode tersebut, untuk mengakses sub property-nya (`Address`, `Affiliation`, dan `GetAffiliationDetailInfo`), tidak perlu dituliskan dari objek terluar, cukup langsung nama property-nya. Sebagai contoh `.Address` di atas merujuk ke variabel `.Info`.



## B.6.7. Seleksi Kondisi

Seleksi kondisi juga bisa dilakukan pada template view. Keyword actions yang digunakan adalah `if` dan `eq` (equal atau sama dengan).

```
 {{if eq .Name "Bruce Wayne"}}
<tr>
  <td colspan="2" style="font-weight: bold;">
    I'm the Batman!
  </td>
</tr>
{{end}}
```



Untuk seleksi kondisi dengan jumlah kondisi lebih dari satu, bisa gunakan `else if`.

```
 {{if pipeline}}
  a
{{else if pipeline}}
  b
{{else}}
  c
{{end}}
```

Untuk seleksi kondisi yang kondisinya adalah bersumber dari variabel bertipe `bool`, maka langsung saja tulis tanpa menggunakan `eq`. Jika kondisi yang diinginkan adalah kebalikan dari nilai variabel, maka gunakan `ne`. Contohnya bisa dilihat pada kode berikut.

```
 {{if .IsTrue}}
  <p>true</p>
{{end}}

{{.isTrue := true}}

{{if isTrue}}
  <p>true</p>
{{end}}

{{if eq isTrue}}
  <p>true</p>
{{end}}

{{if ne isTrue}}
```

```
<p>not true (false)</p>
{{end}}
```

## B.7. Template: Functions

Golang menyediakan beberapa predefiend function yang bisa digunakan dalam file template. Pada bab ini kita akan membahas beberapa diantaranya beserta cara penggunaannya. Cara pemanggilan fungsi atau method sebuah objek pada file template sedikit berbeda dibanding seperti pada bab sebelumnya.

### B.7.1. Persiapan

Siapkan folder proyek baru, dengan isi 2 buah file: `main.go` dan `view.html`. Di dalam file main siapkan sebuah struct berisikan 3 buah property dan 1 method.

```
package main

import "net/http"
import "fmt"
import "html/template"

type Superhero struct {
    Name      string
    Alias     string
    Friends   []string
}

func (s Superhero) SayHello(from string, message string) string {
    return fmt.Sprintf("%s said: \"%s\"", from, message)
}
```

Struct `Superhero` di atas nantinya digunakan untuk mencetak objek yang kemudian disisipkan ke template view.

Selanjutnya buat fungsi `main()`, isi dengan handler untuk rute `/`. Secara umum isi dari file `main.go` ini mirip seperti yang ada pada bab sebelumnya.

```
func main() {
    http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
        var person = Superhero{
            Name:      "Bruce Wayne",
            Alias:     "Batman",
            Friends:   []string{"Superman", "Flash", "Green Lantern"},
        }

        var tmpl = template.Must(template.ParseFiles("view.html"))
        if err := tmpl.Execute(w, person); err != nil {
            http.Error(w, err.Error(), http.StatusInternalServerError)
        }
    })

    fmt.Println("server started at localhost:9000")
    http.ListenAndServe(":9000", nil)
}
```

Kemudian isi file `view.html` dengan kode berikut.

```
<html>
<head>
    <title>Learning html/template Functions</title>
</head>
<body>
</body>
```

```
</html>
```

Jalankan program, buka <http://localhost:9000/>, lalu lanjutkan dengan mengikuti petunjuk di bawah ini.

## B.7.2. Fungsi Escape String

Fungsi pertama yang akan kita bahas adalah `html`. Fungsi ini digunakan untuk meng-escape string. Agar lebih mudah dipahami silakan praktikan kode di bawah ini.

Tulis kode berikut dalam `<body></body>` file `view.html`.

```
<p>
  {{html "<h2>Hello</h2>"}}
</p>
```

Test output yang dihasilkan di browser dengan cukup me-refresh halaman. Tulisan `<h2>Hello</h2>` akan di-escape, dimunculkan sebagai text.



Bisa dilihat bahwa cara untuk menggunakan fungsi pada file template, adalah cukup dengan menuliskan nama fungsinya dalam notasi `{{namaFungsi}}`. Jika fungsi tersebut membutuhkan parameter (seperti fungsi `html`), maka parameternya dituliskan tepat setelah nama fungsi dengan pembatas spasi.

```
{{namaFungsi param1 param2 param3 param4}}
```

Selain fungsi `html`, ada juga beberapa fungsi lain yang sudah disediakan oleh golang.

- Fungsi `js` digunakan untuk meng-escape string `javascript`
- Fungsi `urlquery` digunakan untuk meng-escape string url query

## B.7.3. Fungsi Operator Perbandingan

Pada bab sebelumnya telah dibahas bagaimana penggunaan operator `ne` pada actions `if`. `eq` dan `ne` adalah contoh dari fungsi operator perbandingan. Jika digunakan pada seleksi kondisi yang nilai kondisinya bertipe `bool`, maka cukup dengan menuliskannya setelah operator, contohnya.

```
{{if eq true}}
  benar
{{end}}
```

Nilai kondisi yang bertipe bool hanya bisa digunakan pada `eq` dan `ne` saja

Jika nilai kondisinya merupakan perbandingan, maka nilai yang dibandingkan harus dituliskan, sebagai contoh di bawah ini adalah seleksi kondisi memanfaatkan operator `gt` untuk deteksi apakah nilai di atas 60.

```
{{if gt $value 60}}
  lulus
{{end}}
```

Pada kode di atas, nilai variabel `$value` akan dibandingkan dengan angka `60`, apakah nilainya lebih besar atau tidak.

`gt` merupakan kependekan dari **greater than**

Praktekan kode berikut, tulis ke dalam file `view.html`.

```
{{if eq .Name "Bruce Wayne"}}
  <p>I'm the Batman!</p>
{{else if ne .Name "Clark Kent"}}
  <p>I'm neither Batman or Superman</p>
{{end}}
```

Lihat hasilnya pada browser.



Berikut merupakan daftar operator perbandingan yang didukung oleh template view.

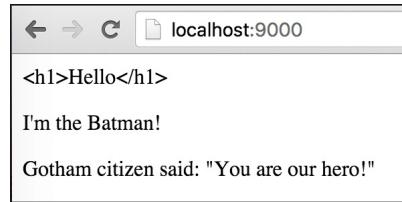
Operator	Penjelasan	Analogi
<code>eq</code>	<i>equal</i> , sama dengan	<code>a == b</code>
<code>ne</code>	<i>not equal</i> , tidak sama dengan	<code>a != b</code>
<code>lt</code>	<i>lower than</i> , lebih kecil	<code>a &lt; b</code>
<code>le</code>	<i>lower than or equal</i> , lebih kecil atau sama dengan	<code>a &lt;= b</code>
<code>gt</code>	<i>greater than</i> , lebih besar	<code>a &gt; b</code>
<code>ge</code>	<i>greater than or equal</i> , lebih besar atau sama dengan	<code>a &gt;= b</code>

## B.7.4. Pemanggilan Method

Cara memanggil method yang disisipkan ke view sama dengan cara pemanggilan fungsi, hanya saja perlu ditambahkan tanda titik `.` (menyesuaikan scope variabelnya). Contohnya bisa dilihat seperti pada kode berikut.

```
<p>
  {{.SayHello "Gotham citizen" "You are our hero!"}}
</p>
```

Test hasilnya pada browser.



## B.7.5. Fungsi String

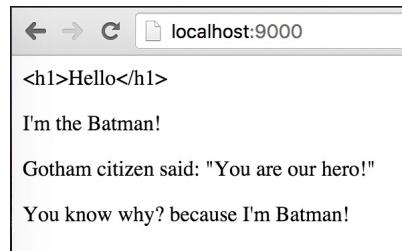
Golang juga menyediakan beberapa fungsi string yang bisa dimanfaatkan, yaitu:

- `print` (merupakan alias dari `fmt.Sprint`)
- `printf` (merupakan alias dari `fmt.Sprintf`)
- `println` (merupakan alias dari `fmt.Println`)

Cara penggunannya juga masih sama.

```
<p>
{{printf "%s because I'm %s" "You know why?" "Batman!"}}
</p>
```

Output:



Jika merasa sedikit bingung memahami statement di atas, mungkin analogi berikut cukup membantu.

```
// template view
printf "%s because I'm %s" "You know why?" "Batman!"

// golang
fmt.Sprintf("%s because I'm %s", "You know why?", "Batman!")
```

Kedua statement di atas menghasilkan output yang sama.

## B.7.6. Fungsi `len` dan `index`

Kegunaan dari fungsi `len` seperti yang sudah diketahui adalah untuk menghitung jumlah elemen. Sedangkan fungsi `index` digunakan jika elemen tertentu ingin diakses.

Sebagai contoh, `Friends` yang merupakan array, diakses elemen indeks ke-1 menggunakan `index`, maka caranya:

```
{{index .Friends 1}}
```

Berikut merupakan contoh penerapan fungsi `len` dan `index`.

```
<p>
Batman have many friends. {{len .Friends}} of them are:
{{index .Friends 0}},
{{index .Friends 1}}, and
{{index .Friends 2}}
</p>
```

Output:

```
<h1>Hello</h1>
I'm the Batman!
Gotham citizen said: "You are our hero!"
You know why? because I'm Batman!
Batman have many friends. 3 of them are: Superman, Flash, and Green Lantern
```

## B.7.7. Fungsi Operator Logika

Selain fungsi operator perbandingan, terdapat juga operator logika `or`, `and`, dan `not`. Cara penggunaannya adalah dengan dituliskan setelah actions `if` atau `elseif`, sebagai fungsi dengan parameter adalah nilai yang ingin dibandingkan.

Fungsi `not` ekuivalen dengan `ne`

```
 {{$cond1 := true}}
 {{$cond2 := false}}

 {{if or $cond1 $cond2}}
     <p>Be like Batman!</p>
 {{end}}
```

Output:

```
<h1>Hello</h1>
I'm the Batman!
Gotham citizen said: "You are our hero!"
You know why? because I'm Batman!
Batman have many friends. 3 of them are: Superman, Flash, and Green Lantern
Be like Batman!
```

## B.8. Template: Custom Functions

Pada bab sebelumnya kita telah mengenal beberapa predefined fungsi yang disediakan oleh Golang. Kali ini kita akan belajar tentang fungsi custom, bagaimana cara membuat dan menggunakannya dalam template.

### B.8.1. Front End

Pertama, siapkan projek baru. Buat file template `view.html`, lalu isi dengan kode berikut.

```
<html>
  <head>
    <title>Learning html/template Functions</title>
  </head>
  <body>
    {{unescape "<!-- this is comment -->"}}
    {{unescape "<h2>"}}
    {{avg 8 9 8 6 7 8 8}}
    {{"</h2>" | unescape}}
  </body>
</html>
```

Ada 2 hal yang perlu diperhatikan dari kode di atas. Pertama, terdapat dua buah fungsi yang dipanggil beberapa kali.

1. Fungsi `unescape()`, digunakan untuk menampilkan string tanpa di-escape
2. Fungsi `avg()`, digunakan untuk mencari rata-rata dari angka-angka yang disisipkan sebagai parameter

Kedua fungsi tersebut adalah fungsi kustom yang akan kita buat.

Hal ke-2, terdapat 1 baris statement yang penulisannya agak unik, yaitu `{{"</h2>" | unescape}}`. Statement tersebut maknanya adalah string `"</h2>"` digunakan sebagai parameter dalam pemanggilan fungsi `unescape`. Tanda pipe atau `|` adalah penanda bahwa parameter dituliskan terlebih dahulu sebelum nama fungsi nya.

### B.8.2. Back End

View sudah siap, sekarang saatnya pindah ke bagian back end. Isi `main.go`, tentukan package sebagai `main` dan import package lain yang diperlukan.

```
package main

import "net/http"
import "fmt"
import "html/template"
```

Selanjutnya beberapa fungsi akan dibuat, lalu disimpan dalam `template.FuncMap`. Pembuatan fungsi dituliskan dalam bentuk key-value atau hash map. Nama fungsi sebagai key, dan body fungsi sebagai value.

```
var funcMap = template.FuncMap{
  "unescape": func(s string) template.HTML {
    return template.HTML(s)
  },
  "avg": func(n ...int) int {
    var total = 0
    for _, each := range n {
      total += each
    }
  }
}
```

```

        return total / len(n)
    },
}

```

template.FuncMap sebenarnya merupakan alias dari `map[string]interface{}`

Dalam `funcMap` di atas, dua buah fungsi disiapkan, `unescape()` dan `avg()`. Nantinya fungsi ini kita gunakan di view.

Setelah itu, siapkan fungsi `main()` dengan isi route handler untuk `/`. Di dalam handler ini, `view.html` diparsing, kemudian disisipkan fungsi yang telah dibuat di atas kedalamnya.

```

func main() {
    http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
        var tmpl = template.Must(template.New("view.html").
            Funcs(funcMap).
            ParseFiles("view.html"))
        if err := tmpl.Execute(w, nil); err != nil {
            http.Error(w, err.Error(), http.StatusInternalServerError)
        }
    })

    fmt.Println("server started at localhost:9000")
    http.ListenAndServe(":9000", nil)
}

```

Berikut merupakan penjelasan step-by-step mengenai kode panjang untuk parsing dan rendering template di atas.

1. Sebuah template disiapkan dengan nama `view.html`. Pembuatan instance template dilakukan melalui fungsi `template.New()`.
2. Fungsi custom yang telah kita buat, diregistrasikan agar dikenali oleh template tersebut. Bisa dilihat pada pemanggilan method `Funcs()`.
3. Setelah itu, lewat method `ParseFiles()`, view `view.html` di-parsing. Akan dicari dalam file tersebut apakah ada template yang didefinisikan dengan nama `view.html`. Karena di dalam template view tidak ada deklarasi template sama sekali (`{{template "namatemplate"}}`), maka akan dicari view yang namanya adalah `view.html`. Keseluruhan isi `view.html` akan dianggap sebagai sebuah template dengan nama template adalah nama file itu sendiri.

### B.8.3. Test

Tes hasilnya lewat browser.



### B.8.4. Perbedaan Fungsi `template.ParseFiles()` & Method `ParseFiles()` Milik `*template.Template`

Pada kode di atas, pemanggilan `template.New()` menghasilkan objek bertipe `*template.Template`.

Pada bab [1.5](#) kita telah belajar mengenai fungsi `template.ParseFiles()`, yang fungsi tersebut juga mengembalikan objek bertipe `*template.Template`.

Pada kode di atas, method `ParseFiles()` yang dipanggil bukanlah fungsi `template.ParseFiles()` yang kita telah pelajari sebelumnya. Meskipun namanya sama, kedua fungsi/method ini berbeda.

- Fungsi `template.ParseFiles()`, adalah milik package `template`. Fungsi ini digunakan untuk mem-parsing semua view yang disisipkan sebagai parameter.
- Method `ParseFiles()`, milik `*template.Template`, digunakan untuk memparsing semua view yang disisipkan sebagai parameter, lalu diambil hanya bagian yang nama template-nya adalah sama dengan nama template yang sudah di-alokasikan menggunakan `template.New()`. Jika template yang dicari tidak ada, maka akan mencari yang nama file-nya sama dengan nama template yang sudah ter-alokasi.

Bab selanjutnya akan membahas lebih detail mengenai penggunaan method `ParseFiles()`.

## B.9. Template: Render Specific HTML Template

Pada bab ini, kita akan belajar bagaimana cara untuk render template html tertentu. Sebuah file view bisa berisikan banyak template. Template mana yang ingin di-render bisa ditentukan.

### B.9.1. Front End

Siapkan folder projek baru, buat file template bernama `view.html`, lalu isi dengan kode berikut.

```
{{define "index"}}
<html>
  <head>
    <title>Learning html/template Functions</title>
  </head>
  <body>
    <h2>Index</h2>
  </body>
</html>
{{end}}

{{define "test"}}
<html>
  <head>
    <title>Other Template</title>
  </head>
  <body>
    <h2>Test</h2>
  </body>
</html>
{{end}}
```

Pada file view di atas, terlihat terdapat 2 template didefinisikan dalam 1 file, template `index` dan `test`. Rencananya template `index` akan ditampilkan ketika rute `/` diakses, dan template `test` ketika rute `/test` diakses.

### B.9.2. Back End

Selanjutnya siapkan back end program, buat file `main.go`, tulis kode berikut.

```
package main

import "net/http"
import "fmt"
import "html/template"

func main() {
    http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
        var tmpl = template.Must(template.New("index").ParseFiles("view.html"))
        if err := tmpl.Execute(w, nil); err != nil {
            http.Error(w, err.Error(), http.StatusInternalServerError)
        }
    })

    http.HandleFunc("/test", func(w http.ResponseWriter, r *http.Request) {
        var tmpl = template.Must(template.New("test").ParseFiles("view.html"))
        if err := tmpl.Execute(w, nil); err != nil {
            http.Error(w, err.Error(), http.StatusInternalServerError)
        }
    })
}
```

```
fmt.Println("server started at localhost:9000")
http.ListenAndServe(":9000", nil)
}
```

Pada kode di atas bisa dilihat, terdapat 2 rute yang masing-masing memparsing file yang sama, tapi spesifik template yang dipilih untuk di-render berbeda.

Contoh di ruta `/`, sebuah template dialokasikan dengan nama `index`, kemudian di-parsing-lah view bernama `view.html` menggunakan method `ParseFiles()`. Golang secara cerdas akan melakukan mencari dalam file view tersebut, apakah ada template yang namanya adalah `index` atau tidak. Jika ada akan ditampilkan. Hal ini juga berlaku pada ruta `/test`, jika isi dari template bernama `test` akan ditampilkan tiap kali ruta tersebut diakses.

### B.9.3. Test

Lakukan tes pada program yang telah kita buat, kurang lebih hasilnya seperti pada gambar berikut.



## B.10. Template: Render HTML String

Output HTML yang muncul, selain bersumber dari template view, bisa juga bersumber dari sebuah string. Dengan menggunakan method `Parse()` milik `*template.Template` kita bisa menjadikan string html sebagai output.

### B.10.1. Praktek

Langsung saja kita praktikkan, siapkan folder projek baru beserta file `main.go`, isi dengan kode berikut.

```
package main

import "net/http"
import "fmt"
import "html/template"

const view string = `<html>
    <head>
        <title>Template</title>
    </head>
    <body>
        <h1>Hello</h1>
    </body>
</html>`
```

Konstanta bernama `view` bertipe `string` disiapkan, dengan isi adalah string html yang akan kita jadikan sebagai output nantinya.

Kemudian buat fungsi `main()`, isinya adalah route handler `/index`. Dalam handler tersebut, string html `view` diparsing lalu dirender sebagai output.

Tambahkan juga route `/`, yang isinya adalah me-redirect request secara paksa ke `/index` menggunakan fungsi `http.Redirect()`.

```
func main() {
    http.HandleFunc("/index", func(w http.ResponseWriter, r *http.Request) {
        var tmpl = template.Must(template.New("main-template").Parse(view))
        if err := tmpl.Execute(w, nil); err != nil {
            http.Error(w, err.Error(), http.StatusInternalServerError)
        }
    })

    http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
        http.Redirect(w, r, "/index", http.StatusTemporaryRedirect)
    })
    fmt.Println("server started at localhost:9000")
    http.ListenAndServe(":9000", nil)
}
```

Pada kode di atas bisa dilihat, sebuah template bernama `main-template` disiapkan. Template tersebut diisi dengan hasil parsing string html `view` lewat method `Parse()`.

### B.10.2. Test

Lakukan tes dan lihat hasilnya.



## B.11. HTTP Method: POST & GET

Setelah sebelumnya kita telah mempelajari banyak hal yang berhubungan dengan template view, kali ini topik yang terpilih sedikit berbeda, yaitu mengenai penanganan http request di back end.

Sebuah route handler pada dasarnya bisa menerima segala jenis request dalam artian apapun HTTP method-nya. Seperti **POST**, **GET**, dan atau lainnya. Untuk memisah request berdasarkan method-nya, bisa menggunakan seleksi kondisi.

Pada bab lain kita akan belajar teknik routing yg lebih advance dengan bantuan routing library.

### B.11.1. Praktek

Silakan pelajari dan praktekkan kode berikut.

```
package main

import "net/http"
import "fmt"

func main() {
    http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
        if r.Method == "POST" {
            w.Write([]byte("post"))
        } else if r.Method == "GET" {
            w.Write([]byte("get"))
        } else {
            http.Error(w, "", http.StatusBadRequest)
        }
    })

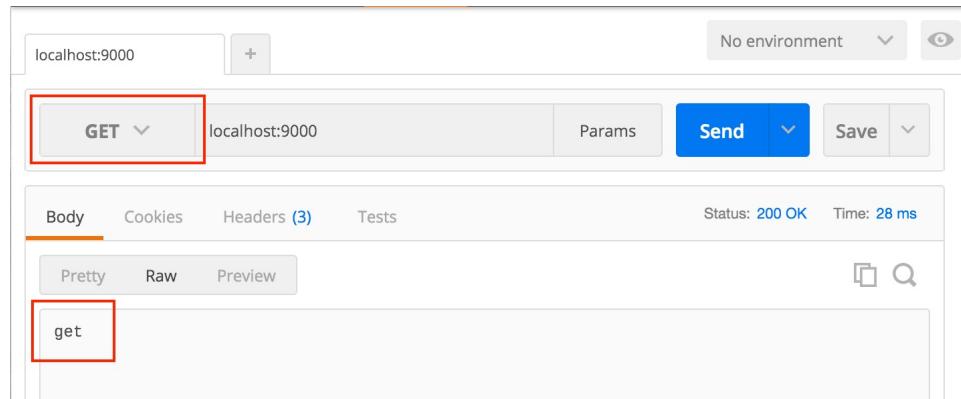
    fmt.Println("server started at localhost:9000")
    http.ListenAndServe(":9000", nil)
}
```

Struct `*http.Request` memiliki property bernama `Method` yang bisa digunakan untuk mengecek method daripada request yang sedang berjalan.

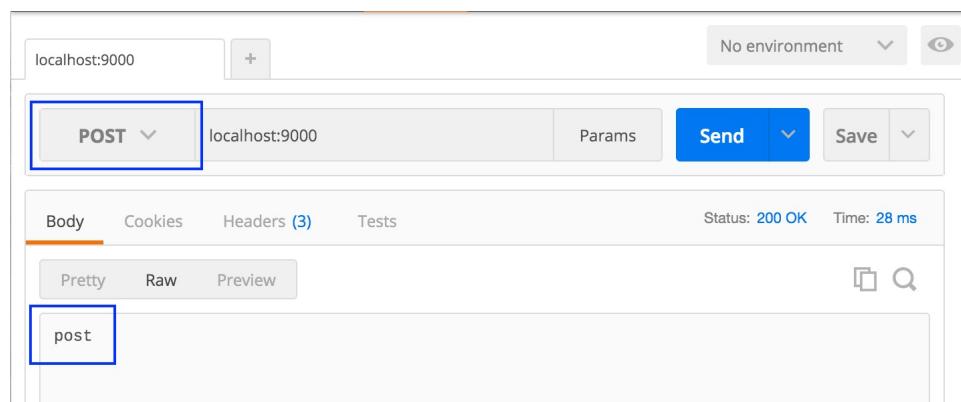
Pada contoh di atas, request ke route `/` dengan method POST akan menghasilkan output text `post`, sedangkan method GET menghasilkan output text `get`.

### B.11.2. Test

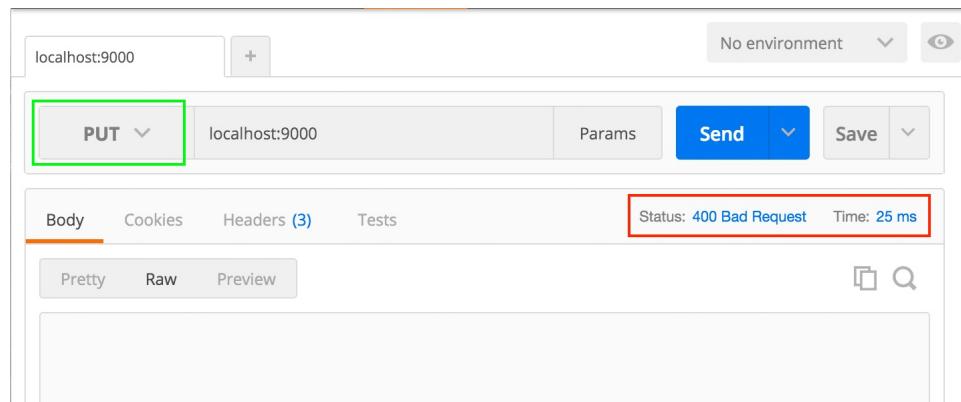
Gunakan [postman](#), atau tools sejenisnya untuk mempermudah testing. Berikut adalah contoh request dengan method GET.



Sedangkan di bawah ini adalah untuk method POST.



Jika method yang digunakan adalah selain POST dan GET, maka sesuai source code di atas, harusnya request akan menghasilkan response **400 Bad Request**. Di bawah ini adalah contoh request dengan method **PUT**.



## B.12. Form Value

Pada bab ini kita akan belajar bagaimana cara untuk submit data, dari form di layer front end, ke back end.

### B.12.1. Front End

Pertama siapkan folder projek baru, dan sebuah file template view `view.html`. Pada file ini perlu didefinisikan 2 buah template, yaitu `form` dan `result`. Template pertama (`form`) dijadikan landing page program, isinya beberapa inputan untuk submit data.

```
{{define "form"}}
<!DOCTYPE html>
<html>
  <head>
    <title>Input Message</title>
  </head>
  <body>
    <form method="post" action="/process">
      <label>Name :</label>
      <input type="text" placeholder="Type name here" name="name" required />
      <br />

      <label>Message :</label>
      <input type="text" placeholder="Type message here" name="message" required />
      <br />

      <button type="submit">Print</button>
    </form>
  </body>
</html>
{{end}}
```

Aksi dari form di atas adalah `/process`, yang dimana url tersebut nantinya akan mengembalikan output berupa html hasil render template `result`. Silakan tulis template `result` berikut dalam `view.html` (jadi file `view` ini berisi 2 buah template).

```
{{define "result"}}
<!DOCTYPE html>
<html>
  <head>
    <title>Show Message</title>
  </head>
  <body>
    <h1>Hello {{.name}}</h1>
    <p>{{.message}}</p>
  </body>
</html>
{{end}}
```

### B.12.2. Back End

Buat file `main.go`. Dalam file ini 2 buah route handler diregistrasikan.

- Route `/` adalah landing page, menampilkan form input.
- Route `/process` sebagai action dari form input, menampilkan text.

```

package main

import "net/http"
import "fmt"
import "html/template"

func main() {
    http.HandleFunc("/", routeIndexGet)
    http.HandleFunc("/process", routeSubmitPost)

    fmt.Println("server started at localhost:9000")
    http.ListenAndServe(":9000", nil)
}

```

Handler route `/` dibungkus dalam fungsi bernama `routeIndexGet`. Di dalamnya, template `form` dalam file template `view.html` akan di-render ke view. Request dalam handler ini hanya dibatasi untuk method GET saja, request dengan method lain akan menghasilkan response 400 Bad Request.

```

func routeIndexGet(w http.ResponseWriter, r *http.Request) {
    if r.Method == "GET" {
        var tmpl = template.Must(template.New("form").ParseFiles("view.html"))
        var err = tmpl.Execute(w, nil)

        if err != nil {
            http.Error(w, err.Error(), http.StatusInternalServerError)
        }
        return
    }

    http.Error(w, "", http.StatusBadRequest)
}

```

Fungsi `routeSubmitPost` yang merupakan handler route `/process`, berisikan proses yang mirip seperti handler route `/`, yaitu parsing `view.html` untuk di ambil template `result` -nya. Selain itu, pada handler ini ada proses pengambilan data yang dikirim dari form ketika di-submit, untuk kemudian disisipkan ke template view.

```

func routeSubmitPost(w http.ResponseWriter, r *http.Request) {
    if r.Method == "POST" {
        var tmpl = template.Must(template.New("result").ParseFiles("view.html"))

        if err := r.ParseForm(); err != nil {
            http.Error(w, err.Error(), http.StatusInternalServerError)
            return
        }

        var name = r.FormValue("name")
        var message = r.Form.Get("message")

        var data = map[string]string{"name": name, "message": message}

        if err := tmpl.Execute(w, data); err != nil {
            http.Error(w, err.Error(), http.StatusInternalServerError)
        }
        return
    }

    http.Error(w, "", http.StatusBadRequest)
}

```

Ketika user submit ke `/process`, maka data-data yang ada di form input dikirim. Method `ParseForm()` pada statement `r.ParseForm()` berguna untuk parsing form data yang dikirim dari view, sebelum akhirnya bisa diambil data-datanya. Method tersebut mengembalikan data `error` jika proses parsing gagal (kemungkinan karena data

yang dikirim ada yang tidak valid).

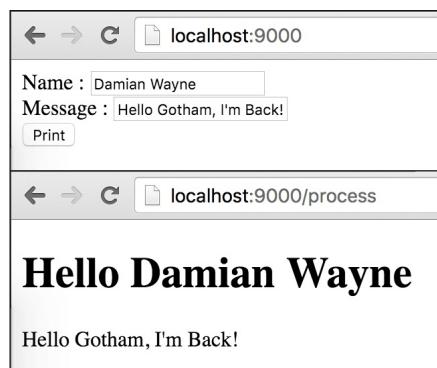
Pengambilan data yang dikirim dilakukan lewat method `FormValue()`. Contohnya seperti pada kode di atas, `r.FormValue("name")`, akan mengembalikan data inputan `name` (data dari inputan `<input name="name" />`).

Selain lewat method `FormValue()`, pengaksesan data juga bisa dilakukan dengan cara mengakses property `Form` terlebih dahulu, kemudian mengakses method `Get()`. Contohnya seperti `r.Form.Get("message")`, yang akan menghasilkan data inputan `message`. Hasil dari kedua cara di atas adalah sama.

Setelah data dari form sudah ditangkap oleh back-end, data ditampung dalam variabel `data` yang bertipe `map[string]string`. Variabel `data` tersebut kemudian disisipkan ke view, lewat statement `tmpl.Execute(w, data)`.

### B.12.3. Test

OK, sekarang coba jalankan program yang telah kita buat, dan cek hasilnya.



## B.13. Form Upload File

Pada bagian ini kita akan belajar bagaimana cara meng-handle upload file lewat form. Caranya beberapa bagian mirip seperti pada bab sebelumnya, hanya perlu ditambahkan proses untuk handling file yang di-upload. File tersebut disimpan ke dalam path/folder tertentu.

### B.13.1. Struktur Folder Proyek

Sebelum mulai masuk ke bagian koding, siapkan terlebih dahulu file dan folder dengan struktur seperti gambar berikut.

Name	Date Modified
files	Today, 6:17 AM
main.go	Feb 25, 2016, 9:36 PM
view.html	Feb 23, 2016, 9:05 AM

Program sederhana yang akan kita buat, memiliki satu form dengan 2 inputan, alias dan file. Data file nantinya disimpan pada folder `files` yang telah dibuat, dengan nama sesuai nama file aslinya. Kecuali ketika user mengisi inputan alias, maka nama tersebut yang akan digunakan sebagai nama file tersimpan.

### B.13.2. Front End

Di bagian front end, isi file `view.html` dengan kode berikut. Template file ini nantinya yang dimunculkan sebagai landing page.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Input Message</title>
  </head>
  <body>
    <form method="post" action="/process" enctype="multipart/form-data">
      <label>The file :</label>
      <input type="file" name="file" required /><br />

      <label>Rename to :</label>
      <input type="text" name="alias" /><br />

      <button type="submit">Submit</button>
    </form>
  </body>
</html>
```

Perlu diperhatikan, pada tag `<form>` perlu ditambahkan atribut `enctype="multipart/form-data"`, agar http request mendukung upload file.

### B.13.3. Back End

Di layer back end ada cukup banyak package yang perlu di-import, seperti `os`, `io`, `path/filepath`, dan lainnya. Packages tersebut kita perlukan untuk handling file upload.

Pada fungsi `main()` siapkan 2 buah route handler, satu untuk landing page, dan satunya lagi digunakan ketika proses upload selesai (sama seperti pada bab sebelumnya).

```
package main

import "net/http"
import "fmt"
import "os"
import "io"
import "path/filepath"
import "html/template"

func main() {
    http.HandleFunc("/", routeIndexGet)
    http.HandleFunc("/process", routeSubmitPost)

    fmt.Println("server started at localhost:9000")
    http.ListenAndServe(":9000", nil)
}
```

Handler route `/` isinya proses untuk menampilkan landing page (file `view.html`). Method yang diperbolehkan mengakses route ini hanya `GET`.

```
func routeIndexGet(w http.ResponseWriter, r *http.Request) {
    if r.Method != "GET" {
        http.Error(w, "", http.StatusBadRequest)
        return
    }

    var tmpl = template.Must(template.ParseFiles("view.html"))
    var err = tmpl.Execute(w, nil)

    if err != nil {
        http.Error(w, err.Error(), http.StatusInternalServerError)
    }
}
```

Selanjutnya siapkan handler untuk route `/process`, yaitu fungsi `routeSubmitPost`. Gunakan statement `r.ParseMultipartForm(1024)` untuk parsing form data yang dikirim.

```
func routeSubmitPost(w http.ResponseWriter, r *http.Request) {
    if r.Method != "POST" {
        http.Error(w, "", http.StatusBadRequest)
        return
    }

    if err := r.ParseMultipartForm(1024); err != nil {
        http.Error(w, err.Error(), http.StatusInternalServerError)
        return
    }

    // ...
}
```

Method `ParseMultipartForm()` digunakan untuk mem-parsing form data yang ada data file nya. Argumen `1024` pada method tersebut adalah `maxMemory`. Pemanggilan method tersebut membuat file yang terupload disimpan sementara pada memory dengan alokasi adalah sesuai dengan `maxMemory`. Jika ternyata kapasitas yang sudah dialokasikan tersebut tidak cukup, maka file akan disimpan dalam temporary file.

Masih dalam fungsi `routeSubmitPost()`, tambahkan kode untuk mengambil data alias dan file.

```

alias := r.FormValue("alias")

uploadedFile, handler, err := r.FormFile("file")
if err != nil {
    http.Error(w, err.Error(), http.StatusInternalServerError)
    return
}
defer uploadedFile.Close()

dir, err := os.Getwd()
if err != nil {
    http.Error(w, err.Error(), http.StatusInternalServerError)
    return
}

```

Statement `r.FormFile("file")` digunakan untuk mengambil file yg di upload, mengembalikan 3 objek:

- Objek bertipe `multipart.File` (yang merupakan turunan dari `*os.File`)
- Informasi header file (bertipe `*multipart.FileHeader`)
- Dan `error` jika ada

Tahap selanjutnya adalah, menambahkan kode membuat file baru, yang nantinya file ini akan diisi dengan isi dari file yang ter-upload. Jika inputan `alias` diisi, maka nama nilai inputan tersebut dijadikan sebagai nama file.

```

filename := handler.Filename
if alias != "" {
    filename = fmt.Sprintf("%s%s", alias, filepath.Ext(handler.Filename))
}

fileLocation := filepath.Join(dir, "files", filename)
targetFile, err := os.OpenFile(fileLocation, os.O_WRONLY|os.O_CREATE, 0666)
if err != nil {
    http.Error(w, err.Error(), http.StatusInternalServerError)
    return
}
defer targetFile.Close()

if _, err := io.Copy(targetFile, uploadedFile); err != nil {
    http.Error(w, err.Error(), http.StatusInternalServerError)
    return
}

w.Write([]byte("done"))

```

Fungsi `filepath.Ext` digunakan untuk mengambil ekstensi dari sebuah file. Pada kode di atas, `handler.Filename` yang berisi nama file terupload diambil ekstensinya, lalu digabung dengan `alias` yang sudah terisi.

Fungsi `filepath.Join` berguna untuk pembentukan path.

Fungsi `os.OpenFile` digunakan untuk membuka file. Fungsi ini membutuhkan 3 buah parameter:

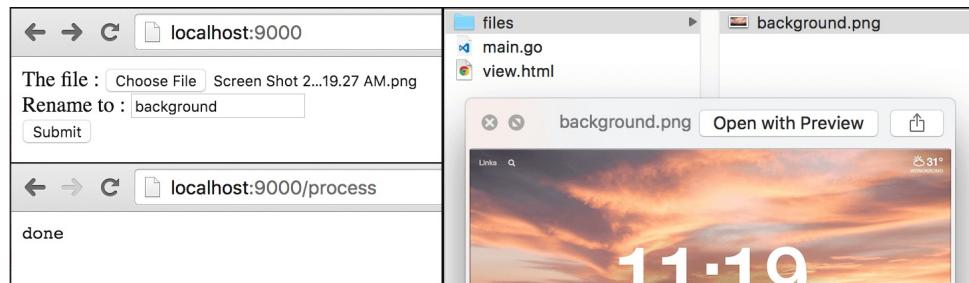
- Parameter pertama merupakan path atau lokasi dari file yang ingin dibuka
- Parameter kedua adalah flag mode, apakah *read only*, *write only*, atau keduanya, atau lainnya.
  - `os.O_WRONLY|os.O_CREATE` maknanya, file yang dibuka hanya bisa ditulis saja (*write only* konsantanya adalah `os.O_WRONLY`), dan file tersebut akan dibuat jika belum ada (konstantanya `os.O_CREATE`).
- Sedangkan parameter terakhir adalah permission dari file, yang digunakan dalam pembuatan file itu sendiri.

Fungsi `io.Copy` akan mengisi konten file parameter pertama (`targetFile`) dengan isi parameter kedua (`uploadedFile`). File kosong yang telah kita buat tadi akan diisi dengan data file yang tersimpan di memory.

Pada bab lain kita akan belajar cara handling file upload dengan metode yang lebih efektif dan hemat memori, yaitu menggunakan `MultipartReader`.

## B.13.4. Testing

Jalankan program, test hasilnya lewat browser.



## B.14. AJAX JSON Payload

Sebelumnya kita telah belajar bagaimana cara submit data dari front-end ke back-end menggunakan teknik **Form Data**. Kali ini kita akan belajar tentang cara request menggunakan teknik **Request Payload** dengan tipe payload adalah **JSON**.

Teknik request **Form Data** digunakan salah satu nya pada request submit lewat `<form />`. Pada bab ini, kita tidak akan menggunakan cara submit lewat form, melainkan menggunakan teknik **AJAX** (Asynchronous JavaScript And XML), dengan payload ber-tipe **JSON**.

**Perbedaan** antara kedua jenis request tersebut adalah pada isi header `Content-Type`, dan bentuk informasi dikirimkan. Secara default, request lewat `<form />`, content type-nya adalah `application/x-www-form-urlencoded`. Data dikirimkan dalam bentuk query string (key-value) seperti `id=n001&nama=bruce`.

Ketika di form ditambahkan atribut `enctype="multipart/form-data"`, maka content type berubah menjadi `multipart/form-data`.

Request Payload JSON sedikit berbeda, `Content-Type` berisikan `application/json`, dan data disisipkan dalam `body` dalam bentuk **JSON** string.

### B.14.1. Struktur Folder Proyek

OK, langsung saja, pertama siapkan proyek dengan struktur seperti pada gambar di bawah ini.

chapter-1.14	Feb 23, 2016, 9:02 AM
assets	Feb 18, 2016, 8:47 AM
jquery-1.12.0.min.js	Feb 18, 2016, 8:46 AM
main.go	Mar 17, 2016, 8:59 AM
view.html	Mar 17, 2016, 8:57 AM

Silakan unduh file js jQuery dari situs official jQuery.

### B.14.2. Front End - HTML

Layout dari view perlu disiapkan terlebih dahulu, tulis kode berikut pada file `view.html`.

```
<!DOCTYPE html>
<html>
  <head>
    <title>JSON Payload</title>
    <script src="static/jquery-1.12.0.min.js"></script>
    <script>
      $(function () {
        // javascript code here
      });
    </script>
  </head>
  <body>
    <p class="message"></p>
    <form id="user-form" method="post" action="/save">
      <!-- html code here -->
    </form>
  </body>
</html>
```

Selanjutnya, pada tag `<form />` tambahkan tabel sederhana berisikan inputan-inputan yang diperlukan. Ada tiga buah inputan yang harus dipersiapkan, yaitu: *Name*, *Age*, dan *Gender*, dan juga sebuah button untuk submit form.

```

<table noborder>
  <tr>
    <td>
      <label>Name :</label>
    </td>
    <td>
      <input required type="text" name="name" placeholder="Type name here" />
    </td>
  </tr>
  <tr>
    <td>
      <label>Age :</label>
    </td>
    <td>
      <input required type="number" name="age" placeholder="Set age" />
    </td>
  </tr>
  <tr>
    <td>
      <label>Gender :</label>
    </td>
    <td>
      <select name="gender" required style="width: 100%;">
        <option value="">Select one</option>
        <option value="male">Male</option>
        <option value="female">Female</option>
      </select>
    </td>
  </tr>
  <tr>
    <td colspan="2" style="text-align: right;">
      <button type="submit">Save</button>
    </td>
  </tr>
</table>

```

### B.14.3. Front End - HTML

Sekarang kita masuk ke bagian paling menyenangkan, yaitu javascript. Siapkan sebuah event `submit` pada `#user-form`. Dalam event tersebut event submit default milik `<form />` di-override, diganti dengan AJAX request.

```

$( "#user-form" ).on("submit", function (e) {
  e.preventDefault();

  var $self = $(this);
  var payload = JSON.stringify({
    name: $('[name="name"]').val(),
    age: parseInt($('.[name="age"]').val(), 10),
    gender: $('[name="gender"]').val()
  });

  $.ajax({
    url: $self.attr("action"),
    type: $self.attr("method"),
    data: payload,
    contentType: 'application/json',
  }).then(function (res) {
    $(".message").text(res);
  }).catch(function (a) {
    alert("ERROR: " + a.responseText);
  });
}

```

```
});
```

Value semua inputan diambil lalu dimasukkan dalam sebuah objek lalu di stringify (agar menjadi JSON string), untuk kemudian dijadikan sebagai payload request. Bisa dilihat pada kode AJAX di atas, `contentType` nilainya adalah `application/json`.

Respon dari ajax di atas akan dimunculkan pada `<p class="message"></p>`.

## B.14.4. Back End

3 buah rute perlu disiapkan, yang pertama adalah untuk menampilkan `view.html`, untuk keperluan submit data, dan registrasi asset.

```
package main

import "fmt"
import "net/http"
import "html/template"
import "encoding/json"

func main() {
    http.HandleFunc("/", handleIndex)
    http.HandleFunc("/save", handleSave)

    http.Handle("/static/",
        http.StripPrefix("/static/",
            http.FileServer(http.Dir("assets"))))

    fmt.Println("server started at localhost:9000")
    http.ListenAndServe(":9000", nil)
}
```

Handler `handleIndex` berisikan kode untuk parsing `view.html`.

```
func handleIndex(w http.ResponseWriter, r *http.Request) {
    tmpl := template.Must(template.ParseFiles("view.html"))
    if err := tmpl.Execute(w, nil); err != nil {
        http.Error(w, err.Error(), http.StatusInternalServerError)
    }
}
```

Sedangkan `handleSave` akan memproses request yang di-submit dari bagian depan.

```
func handleSave(w http.ResponseWriter, r *http.Request) {
    if r.Method == "POST" {
        decoder := json.NewDecoder(r.Body)
        payload := struct {
            Name   string `json:"name"`
            Age    int    `json:"age"`
            Gender string `json:"gender"`
        }{}
        if err := decoder.Decode(&payload); err != nil {
            http.Error(w, err.Error(), http.StatusInternalServerError)
            return
        }

        message := fmt.Sprintf(
            "hello, my name is %.s. I'm %d year old %.s",
            payload.Name,
            payload.Age,
            payload.Gender,
```

```

        )
        w.Write([]byte(message))
        return
    }

    http.Error(w, "Only accept POST request", http.StatusBadRequest)
}

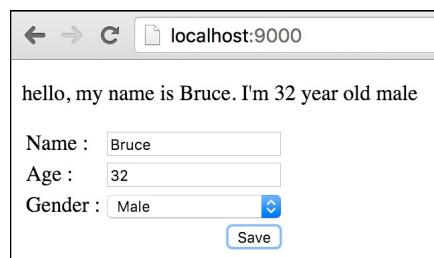
```

Isi payload didapatkan dengan cara men-decode body request (`r.Body`). Proses decoding tidak dilakukan menggunakan `json.Unmarshal()` melainkan lewat json decoder, karena akan lebih efisien untuk jenis kasus seperti ini.

Gunakan `json.Decoder` jika data adalah stream `io.Reader`. Gunakan `json.Unmarshal()` untuk decode data sumbernya sudah ada di memory.

## B.14.5. Test

Jalankan program, test hasilnya di browser.



Gunakan fasilitas Developer Tools pada Chrome untuk melihat detail dari request.

Name	Value
Request Headers	<ul style="list-style-type: none"> <li>Accept: */*</li> <li>Accept-Encoding: gzip, deflate</li> <li>Accept-Language: en-US,en;q=0.8,id;q=0.6</li> <li>Connection: keep-alive</li> <li>Content-Length: 41</li> <li>Content-Type: application/json</li> <li>Cookie: KnotSessionId=R60RQJ81_516K24t98ayZzE2Myi_HcqI</li> <li>Host: localhost:9000</li> <li>Origin: http://localhost:9000</li> <li>Referer: http://localhost:9000/</li> <li>User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_X-Requested-With: XMLHttpRequest</li> </ul>
Request Payload	<pre>{   "name": "Bruce",   "age": 32,   "gender": "male" }</pre>

## B.15. AJAX JSON Response

Pada bab sebelumnya, kita belajar cara untuk memproses request dengan payload JSON string. Pada bab ini kita akan belajar untuk membuat satu endpoint yang mengembalikan data JSON string.

### B.15.1. Praktek

Siapkan satu buah folder proyek baru, dengan satu buah file di dalamnya bernama `main.go`. Dalam file ini siapkan rute `/`.

```
package main

import "fmt"
import "net/http"
import "encoding/json"

func main() {
    http.HandleFunc("/", ActionIndex)

    fmt.Println("server started at localhost:9000")
    http.ListenAndServe(":9000", nil)
}
```

Selanjutnya buat handler untuk rute `/`. Di dalam fungsi ini, data dummy ber-type slice object disiapkan. Data ini akan dikonversi ke JSON lalu dijadikan nilai balik endpoint `/`.

```
func ActionIndex(w http.ResponseWriter, r *http.Request) {
    data := [] struct {
        Name string
        Age  int
    } {
        { "Richard Grayson", 24 },
        { "Jason Todd", 23 },
        { "Tim Drake", 22 },
        { "Damian Wayne", 21 },
    }

    jsonInBytes, err := json.Marshal(data)
    if err != nil {
        http.Error(w, err.Error(), http.StatusInternalServerError)
        return
    }

    w.Header().Set("Content-Type", "application/json")
    w.Write(jsonInBytes)
}
```

Cara mengkonversi data ke bentuk json cukup mudah, bisa menggunakan `json.Marshal()`. Fungsi ini mengembalikan dua nilai balik, data json (dalam bentuk `[]byte`) dan error jika ada.

Untuk mengambil bentuk string dari hasil konversi JSON, cukup lakukan casting pada data slice bytes tersebut. Contoh: `string(jsonInBytes)`

Karena nilai balik konversi sudah dalam bentuk bytes, maka langsung saja panggil method `write()` milik `http.ResponseWriter` dan sisipkan data json sebagai argument.

Jangan lupa juga untuk menambahkan response header `Content-Type: application/json`.

## B.15.2. Testing

OK, semua sudah selesai, lakukan testing.

```

GET http://localhost:9000/ Send 200 OK TIME 7.82 ms SIZE 131 B
Preview 3 Header 3
NAME VALUE
Content-Type application/json
Date Tue, 22 May 2018 08:38:08 GMT
Content-Length 131
Copy to Clipboard
1 [
2 {
3   "Name": "Richard Grayson",
4   "Age": 24
5 },
6 {
7   "Name": "Jason Todd",
8   "Age": 23
9 },
10 {
11   "Name": "Tim Drake",
12   "Age": 22
13 },
14 {
15   "Name": "Damian Wayne",
16   "Age": 21
17 }
18 ]

```

## B.15.3. JSON Response menggunakan JSON.Encoder

Di bab sebelumnya sudah disinggung, bahwa lebih baik menggunakan `json.Decoder` jika ingin men-decode data yang sumbernya ada di stream `io.Reader`

Package json juga memiliki fungsi lainnya yaitu `json.Encoder`, yang sangat cocok digunakan untuk meng-encode data menjadi JSON dengan tujuan objek langsung ke stream `io.Reader`.

Karena tipe `http.ResponseWriter` adalah meng-embed `io.Reader`, maka jelasnya bisa kita terapkan penggunaan encoder di sini.

Contohnya penerapannya sebagai berikut.

```
w.Header().Set("Content-Type", "application/json")

err := json.NewEncoder(w).Encode(data)
if err != nil {
    http.Error(w, err.Error(), http.StatusInternalServerError)
    return
}
```

Kode di atas hasilnya ekuivalen dengan penggunaan `json.Marshal`.

## B.16. AJAX Multiple File Upload

Pada bab ini, kita akan belajar 3 hal dalam satu waktu, yaitu:

1. Bagaimana cara untuk upload file via AJAX.
2. Cara untuk handle upload banyak file sekaligus.
3. Cara handle upload file yang lebih hemat memori.

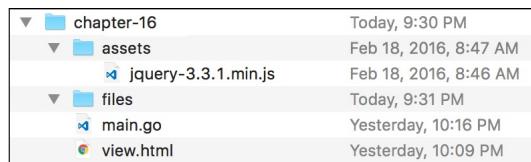
Sebelumnya di bab 13, pemrosesan file upload dilakukan lewat `ParseMultipartForm`, sedangkan pada bab ini metode yang dipakai berbeda, menggunakan `MultipartReader`.

Kelebihan dari `MultipartReader` adalah, file yang di upload **tidak** di simpan sebagai file temporary di lokal terlebih dahulu (tidak seperti `ParseMultipartForm`), melainkan langsung diambil dari stream `io.Reader`.

Di bagian front end, upload file secara asynchronous bisa dilakukan menggunakan objek `FormData`. Semua file dimasukkan dalam `FormData`, lalu objek ini dijadikan payload AJAX request.

### B.16.1. Struktur Folder Proyek

Mari langsung kita praktekkan, pertama siapkan proyek dengan struktur seperti gambar di bawah ini.



Silakan unduh file js jQuery dari situs official jQuery.

### B.16.2. Front End

Buka `view.html`, siapkan template dasar view. Dalam file ini terdapat satu buah inputan upload file yang mendukung multi upload, dan satu buah tombol submit.

Untuk meng-enable kapabilitas multi upload, cukup tambahkan atribut `multiple` pada input file.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Multiple Upload</title>
    <script src="static/jquery-3.3.1.min.js"></script>
    <script>
      $(function () {
        // javascript code goes here
      });
    </script>
  </head>
  <body>
    <form id="user-form" method="post" action="/upload">
      <input required multiple id="upload-file" type="file" />
      <br />
      <button id="btn-upload" type="submit">Upload!</button>
    </form>
  </body>
</html>
```

Override event `submit` pada form `#user-form`, handler event ini berisikan proses mulai pembentukan objek `FormData` dari file-file yang telah di upload, hingga eksekusi AJAX.

```
$("#user-form").on("submit", function (e) {
    e.preventDefault();

    var $self = $(this);
    var files = $("#upload-file")[0].files;
    var formData = new FormData();

    for (var i = 0; i < files.length; i++) {
        formData.append("files", files[i]);
    }

    $.ajax({
        url: $self.attr("action"),
        type: $self.attr("method"),
        data: formData,
        processData: false,
        contentType: false,
    }).then(function (res) {
        alert(res);
        $("#user-form").trigger("reset");
    }).catch(function (a) {
        alert("ERROR: " + a.responseText);
    });
});
```

Objek inputan `files` (yang didapat dari `$("#upload-file")[0].files`) memiliki property `.files` yang isinya merupakan array dari semua file yang dipilih oleh user ketika upload. File-file tersebut di-loop, dimasukkan ke dalam objek `FormData` yang telah dibuat.

AJAX dilakukan lewat `jQuery.ajax`. Berikut adalah penjelasan mengenai konfigurasi `processData` dan `contentType` dalam AJAX yang sudah dibuat.

- Konfigurasi `contentType` perlu di set ke `false` agar header Content-Type yang dikirim bisa menyesuaikan data yang disisipkan.
- Konfigurasi `processData` juga perlu di set ke `false`, agar data yang akan di kirim tidak otomatis dikonversi ke query string atau json string (tergantung `contentType`). Pada konteks ini kita memerlukan payload tetap dalam tipe `FormData`.

## B.16.3. Back End

Ada 2 route handler yang harus dipersiapkan di back end. Pertama adalah rute `/` yang menampilkan form upload, dan rute `/upload` untuk pemrosesan upload sendiri.

Buka file `main.go`, isi dengan package yang dibutuhkan, lalu lakukan registrasi dua rute yang dimaksud di atas, beserta satu buah rute untuk static assets.

```
package main

import "fmt"
import "net/http"
import "html/template"
import "path/filepath"
import "io"
import "os"

func main() {
    http.HandleFunc("/", handleIndex)
    http.HandleFunc("/upload", handleUpload)
```

```

http.Handle("/static/",
    http.StripPrefix("/static/",
        http.FileServer(http.Dir("assets"))))

fmt.Println("server started at localhost:9000")
http.ListenAndServe(":9000", nil)
}

```

Buat handler rute `/`, parsing template view `view.html`.

```

func handleIndex(w http.ResponseWriter, r *http.Request) {
    tmpl := template.Must(template.ParseFiles("view.html"))
    if err := tmpl.Execute(w, nil); err != nil {
        http.Error(w, err.Error(), http.StatusInternalServerError)
    }
}

```

Sebelumnya, pada bab 13, metode yang digunakan untuk handle file upload adalah menggunakan `ParseMultipartForm`, file diproses dalam memori dengan alokasi tertentu, dan jika melebihi alokasi maka akan disimpan pada temporary file.

Metode tersebut kurang tepat guna jika digunakan untuk memproses file yang ukurannya besar (tidak boleh melebihi `maxMemory`) atau jumlah filenya sangat banyak (memakan waktu, karena isi dari masing-masing file akan ditampung pada file `temporary` sebelum benar-benar di-copy ke file tujuan).

Solusinya dari dua masalah di atas adalah menggunakan `MultipartReader` untuk handling file upload. Dengan metode ini, file destinasi isinya akan di-copy langsung dari stream `io.Reader`, tanpa butuh file temporary untuk perantara.

Kembali ke bagian perkodingan, siapkan fungsi `handleUpload`, isinya kode berikut.

```

func handleUpload(w http.ResponseWriter, r *http.Request) {
    if r.Method != "POST" {
        http.Error(w, "Only accept POST request", http.StatusBadRequest)
        return
    }

    basePath, _ := os.Getwd()
    reader, err := r.MultipartReader()
    if err != nil {
        http.Error(w, err.Error(), http.StatusInternalServerError)
        return
    }

    // ...
}

```

Bisa dilihat, method `.MultipartReader()` dipanggil dari objek request milik handler. Mengembalikan dua objek, pertama `*multipart.Reader` dan `error` (jika ada).

Selanjutnya lakukan perulangan terhadap objek `reader`. Setiap file yang di-upload diproses di masing-masing perulangan. Setelah looping berakhir. idealnya semua file sudah terproses dengan benar.

```

for {
    part, err := reader.NextPart()
    if err == io.EOF {
        break
    }

    fileLocation := filepath.Join(basePath, "files", part.FileName())
    dst, err := os.Create(fileLocation)
}

```

```

if dst != nil {
    defer dst.Close()
}
if err != nil {
    http.Error(w, err.Error(), http.StatusInternalServerError)
    return
}

if _, err := io.Copy(dst, part); err != nil {
    http.Error(w, err.Error(), http.StatusInternalServerError)
    return
}

w.Write([]byte(`all files uploaded`))

```

Method `.NextPart()` mengembalikan 2 informasi, yaitu objek stream `io.Reader` (dari file yg di upload), dan `error`.

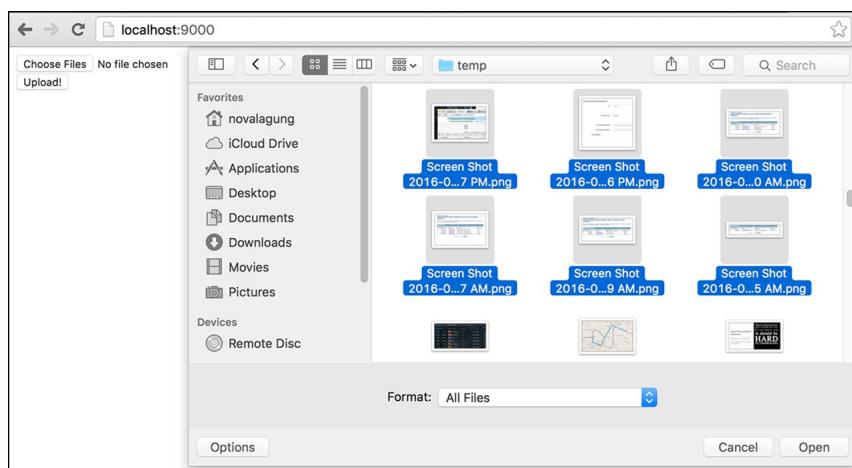
File destinasi dipersiapkan, kemudian diisi dengan data dari stream file, menggunakan `io.Copy()`.

Jika `reader.NextPart()` mengembalikan error `io.EOF`, menandakan bahwa semua file sudah di proses, maka hentikan perulangan.

OK, semua persiapan sudah cukup.

## B.16.4. Testing

Buka browser, test program yang telah dibuat. Coba lakukan pengujian dengan beberapa buah file.



Cek apakah file sudah terupload.





## B.17. Download File

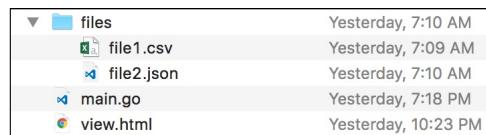
Setelah sebelumnya belajar cara untuk handle upload file, kali ini kita akan belajar bagaimana cara membuat endpoint yang outputnya adalah download file.

Sebenarnya download file bisa dengan mudah di-implementasikan menggunakan fungsi `http.HandleFunc()`, dengan langsung mengakses url dari public assets di browser. Namun outcome dari teknik ini sangat tergantung pada browser. Tiap browser memiliki behaviour berbeda, ada yang file tidak di-download melainkan dibuka di tab, ada yang ter-download.

Dengan menggunakan teknik berikut, file pasti akan ter-download.

### B.17.1. Struktur Folder Proyek

OK, pertama siapkan terlebih dahulu proyek dengan struktur seperti gambar berikut.



File yang berada di folder `files` adalah dummy, jadi anda bisa gunakan file apapun dengan jumlah berapapun untuk keperluan belajar.

### B.17.2. Front End

Kali ini di bagian front end kita tidak menggunakan jQuery, cukup javascript saja tanpa library.

Pertama siapkan dahulu template nya, isi file `view.html` dengan kode berikut.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Download file</title>
    <script>
      // javascript code goes here
    </script>
  </head>
  <body>
    <ul id="list-files"></ul>
  </body>
</html>
```

Tag `<ul />` nantinya akan berisikan list semua file yang ada dalam folder `files`. Data list file didapat dari back end. Diperlukan sebuah AJAX untuk pengambilan data tersebut.

Siapkan sebuah fungsi dengan nama `Yo` atau bisa lainnya, fungsi ini berisikan closure `renderData()`, `getAllListFiles()`, dan method `init()`. Buat instance object baru dari `Yo`, lalu akses method `init()`, tempatkan dalam event `window.onload`.

```
function Yo() {
  var self = this;
  var $ul = document.getElementById("list-files");

  var renderData = function (res) {
```

```
// do stuff
};

var getAllListFiles = function () {
    // do stuff
};

self.init = function () {
    getAllListFiles();
};

window.onload = function () {
    new Yo().init();
};
```

Closure `renderData()` bertugas untuk melakukan rendering data JSON ke HTML. Berikut adalah isi dari fungsi ini.

```
var renderData = function (res) {
    res.forEach(function (each) {
        var $li = document.createElement("li");
        var $a = document.createElement("a");

        $li.innerText = "download ";
        $li.appendChild($a);
        $ul.appendChild($li);

        $a.href = "/download?path=" + encodeURI(each.path);
        $a.innerText = each.filename;
        $a.target = "_blank";
    });
};
```

Sedangkan closure `getAllListFiles()`, memiliki tugas untuk request ke back end, mengambil data list semua file. Request dilakukan dalam bentuk AJAX, nilai baliknya adalah data JSON. Setelah data sudah di tangan, fungsi `renderData()` dipanggil.

```
var getAllListFiles = function () {
    var xhr = new XMLHttpRequest();
    xhr.open("GET", "/list-files");
    xhr.onreadystatechange = function () {
        if (xhr.readyState == 4 && xhr.status == 200) {
            var json = JSON.parse(xhr.responseText);
            renderData(json);
        }
    };
    xhr.send();
};
```

### B.17.3. Back End

Pindah ke bagian back end. Siapkan beberapa hal pada `main.go`, import package, siapkan fungsi main, dan buat beberapa rute.

```
package main

import "fmt"
import "net/http"
import "html/template"
import "path/filepath"
import "io"
```

```

import "encoding/json"
import "os"

type M map[string]interface{}

func main() {
    http.HandleFunc("/", handleIndex)
    http.HandleFunc("/list-files", handleListFiles)
    http.HandleFunc("/download", handleDownload)

    fmt.Println("server started at localhost:9000")
    http.ListenAndServe(":9000", nil)
}

```

Buat handler untuk rute `/`.

```

func handleIndex(w http.ResponseWriter, r *http.Request) {
    tmpl := template.Must(template.ParseFiles("view.html"))
    if err := tmpl.Execute(w, nil); err != nil {
        http.Error(w, err.Error(), http.StatusInternalServerError)
    }
}

```

Lalu siapkan juga route handler `/list-files`. Isi dari handler ini adalah membaca semua file yang ada pada folder `files` untuk kemudian dikembalikan sebagai output berupa JSON. Endpoint ini akan diakses oleh AJAX dari front end.

```

func handleListFiles(w http.ResponseWriter, r *http.Request) {
    files := []M{}
    basePath, _ := os.Getwd()
    filesLocation := filepath.Join(basePath, "files")

    err := filepath.Walk(filesLocation, func(path string, info os.FileInfo, err error) error {
        if err != nil {
            return err
        }

        if info.IsDir() {
            return nil
        }

        files = append(files, M{"filename": info.Name(), "path": path})
        return nil
    })
    if err != nil {
        http.Error(w, err.Error(), http.StatusInternalServerError)
        return
    }

    res, err := json.Marshal(files)
    if err != nil {
        http.Error(w, err.Error(), http.StatusInternalServerError)
        return
    }

    w.Header().Set("Content-Type", "application/json")
    w.Write(res)
}

```

Fungsi `os.Getwd()` mengembalikan informasi absolute path dimana aplikasi di-eksekusi. Path tersebut kemudian di gabung dengan folder bernama `files` lewat fungsi `filepath.Join`.

Fungsi `filepath.Join` akan menggabungkan item-item dengan path separator sesuai dengan sistem operasi dimana program dijalankan. \ untuk Wind\*ws dan / untuk \*nix.

Fungsi `filepath.Walk` berguna untuk membaca isi dari sebuah direktori, apa yang ada didalamnya (file maupun folder) akan di-loop. Dengan memanfaatkan callback parameter kedua fungsi ini (yang bertipe `filepath.WalkFunc`), kita bisa mengambil informasi tiap item satu-per satu.

Selanjutnya siapkan handler untuk `/download`. Implementasi teknik download pada dasarnya sama pada semua bahasa pemrograman, yaitu dengan memainkan header **Content-Disposition** pada response.

```
func handleDownload(w http.ResponseWriter, r *http.Request) {
    if err := r.ParseForm(); err != nil {
        http.Error(w, err.Error(), http.StatusInternalServerError)
        return
    }

    path := r.FormValue("path")
    f, err := os.Open(path)
    if err != nil {
        http.Error(w, err.Error(), http.StatusInternalServerError)
        return
    }

    contentDisposition := fmt.Sprintf("attachment; filename=%s", f.Name())
    w.Header().Set("Content-Disposition", contentDisposition)

    if _, err := io.Copy(w, f); err != nil {
        http.Error(w, err.Error(), http.StatusInternalServerError)
        return
    }
}
```

**Content-Disposition** adalah salah satu ekstensi MIME protocol, berguna untuk menginformasikan browser bagaimana dia harus berinteraksi dengan output. Ada banyak jenis value content-disposition, salah satunya adalah attachment . Pada kode di atas, header `Content-Disposition: attachment; filename=filename.json` menghasilkan output response berupa attachment atau file, yang kemudian akan di-download oleh browser.

Objek file yang direpresentasikan variabel `f`, isinya di-copy ke objek response lewat statement `io.Copy(w, f)` .

## B.17.4. Testing

Jalankan program, akses route `/` . List semua file dalam folder `files` muncul di sana. Klik salah satu file untuk men-download-nya.



## B.18. HTTP Basic Authentication

HTTP Basic Auth adalah salah satu teknik otentifikasi http request. Metode ini membutuhkan informasi username dan password untuk disisipkan dalam header request (dengan format tertentu), jadi cukup sederhana, tidak memerlukan cookies maupun session. Lebih jelasnya silakan baca [RFC-7617](#).

Informasi username dan password tidak serta merta disisipkan dalam header, informasi tersebut harus di-encode terlebih dahulu ke dalam format yg sudah ditentukan sesuai spesifikasi, lalu dimasukan ke header.

Berikut adalah contoh penulisan basic auth.

```
// Request header
Authorization: Basic c29tZXVzZXJuYW1lOnNvbWwYXNzd29yZA==
```

Informasi disisipkan dalam request header dengan key `Authorization`, dan value adalah `Basic` spasi hasil enkripsi dari data username dan password. Data username dan password digabung dengan separator tanda titik dua ( : ), lalu di-encode dalam format encoding Base 64.

```
// Username password encryption
base64encode("someusername:somepassword")
// Hasilnya adalah c29tZXVzZXJuYW1lOnNvbWwYXNzd29yZA==
```

Golang menyediakan fungsi untuk meng-handle request basic auth dengan cukup mudah, jadi tidak perlu untuk memarsing header request terlebih dahulu untuk mendapatkan informasi username dan password.

### B.18.1. Struktur Folder Proyek dan Endpoint

Ok, mari kita praktekan. Pada bab ini kita akan membuat sebuah web service sederhana, isinya satu buah endpoint. Endpoint ini kita manfaatkan sebagai dua endpoint, dengan pembeda adalah informasi pada query string-nya.

- Endpoint `/student` , menampilkan semua data siswa.
- Endpoint `/student?id=s001` , menampilkan data siswa sesuai dengan id yang di minta.

Data siswa sendiri merupakan slice object yang disimpan di variabel global.

OK, langsung saja kita praktekan. Siapkan 3 buah file berikut, tempatkan dalam satu folder proyek.



### B.18.2. Routing

Buka `main.go` , isi dengan kode berikut.

```
package main

import "net/http"
import "fmt"
import "encoding/json"
```

```

func main() {
    http.HandleFunc("/student", ActionStudent)

    server := new(http.Server)
    server.Addr = ":9000"

    fmt.Println("server started at localhost:9000")
    server.ListenAndServe()
}

```

Siapkan handler untuk rute `/student`.

```

func ActionStudent(w http.ResponseWriter, r *http.Request) {
    if !Auth(w, r) { return }
    if !AllowOnlyGET(w, r) { return }

    if id := r.URL.Query().Get("id"); id != "" {
        OutputJSON(w, SelectStudent(id))
        return
    }

    OutputJSON(w, GetStudents())
}

```

Di dalam rute `/student` terdapat beberapa validasi.

- Validasi `!Auth(w, r)`; Nantinya akan kita buat fungsi `Auth()` untuk mengecek apakah request merupakan valid basic auth request atau tidak.
- Validasi `!AllowOnlyGET(w, r)`; Nantinya juga akan kita siapkan fungsi `AllowOnlyGET()`, gunanya untuk memastikan hanya request dengan method `GET` yang diperbolehkan masuk.

Setelah request lolos dari 2 validasi di atas, kita cek lagi apakah request ini memiliki parameter student id.

- Ketika tidak ada parameter student id, maka endpoint ini mengembalikan semua data user yang ada, lewat pemanggilan fungsi `GetStudents()`.
- Sedangkan jika ada parameter student id, maka hanya user dengan id yg diinginkan yg dijadikan nilai balik, lewat fungsi `SelectStudent(id)`.

Selanjutnya tambahkan satu fungsi lagi di main, `OutputJSON()`. Fungsi ini digunakan untuk mengkonversi data menjadi JSON string.

```

func OutputJSON(w http.ResponseWriter, o interface{}) {
    res, err := json.Marshal(o)
    if err != nil {
        w.Write([]byte(err.Error()))
        return
    }

    w.Header().Set("Content-Type", "application/json")
    w.Write(res)
    w.Write([]byte("\n"))
}

```

Konversi dari objek atau slice ke JSON string bisa dilakukan dengan memanfaatkan `json.Marshal`. Untuk lebih jelasnya silakan baca [Dasar Pemrograman Golang, Bab 50, tentang JSON](#).

### B.18.3. Data Student

Buka file `student.go`, siapkan struct `Student` dan variabel untuk menampung data yang bertipe `[]Student`. Data inilah yang dijadikan nilai balik di endpoint yang sudah dibuat.

```
package main

var students = []*Student{}

type Student struct {
    Id   string
    Name string
    Grade int32
}
```

Buat fungsi `GetStudents()`, fungsi ini mengembalikan semua data student. Dan buat juga fungsi `SelectStudent(id)`, fungsi ini mengembalikan data student sesuai dengan id terpilih.

```
func GetStudents() []*Student {
    return students
}

func SelectStudent(id string) *Student {
    for _, each := range students {
        if each.Id == id {
            return each
        }
    }

    return nil
}
```

*Last but not least*, implementasikan fungsi `init()`, buat beberapa dummy data untuk ditampung pada variabel `students`.

Fungsi `init()` adalah fungsi yang secara otomatis dipanggil ketika package-dimana-fungsi-ini-berada di-import atau di run.

```
func init() {
    students = append(students, &Student{Id: "s001", Name: "bourne", Grade: 2})
    students = append(students, &Student{Id: "s002", Name: "ethan", Grade: 2})
    students = append(students, &Student{Id: "s003", Name: "wick", Grade: 3})
}
```

## B.18.4. Fungsi `Auth()` dan `AllowOnlyGET()`

Selanjutnya, kita perlu menyiapkan beberapa fungsi yg digunakan pada `main.go`, yaitu `Auth()` dan `AllowOnlyGET()`.

- **Fungsi `Auth()`**

Buka `middleware.go`, siapkan fungsi `Auth()`.

```
package main

import "net/http"

const USERNAME = "batman"
const PASSWORD = "secret"

func Auth(w http.ResponseWriter, r *http.Request) bool {
    username, password, ok := r.BasicAuth()
```

```

if !ok {
    w.Write([]byte(`something went wrong`))
    return false
}

isValid := (username == USERNAME) && (password == PASSWORD)
if !isValid {
    w.Write([]byte(`wrong username/password`))
    return false
}

return true
}

```

Tugas fungsi `Auth()` adalah memvalidasi apakah request merupakan valid basic auth request, dan juga apakah credentials yang dikirim cocok dengan data pada aplikasi kita. Informasi acuan credentials sendiri di hardcode pada konstanta `USERNAME` dan `PASSWORD`.

Fungsi `r.BasicAuth()` mengembalikan 3 informasi:

1. Username
2. Password
3. Nilai balik ke-3 ini adalah representasi valid tidaknya basic auth request yang sedang berlangsung

Jika basic auth request tidak valid, maka tampilkan pesan error sebagai nilai balik. Sedangkan jika basic auth adalah valid, maka dilanjutkan ke proses otentikasi, mengecek apakah username dan password yang dikirim cocok dengan username dan password yang ada di aplikasi kita.

### • Fungsi `AllowOnlyGET()`

Fungsi ini bertugas untuk memastikan bahwa request yang diperbolehkan hanya yang ber-method `GET`. Selainnya, maka akan dianggap invalid request.

```

func AllowOnlyGET(w http.ResponseWriter, r *http.Request) bool {
    if r.Method != "GET" {
        w.Write([]byte("Only GET is allowed"))
        return false
    }

    return true
}

```

## B.18.5. Testing

Semuanya sudah siap, jalankan aplikasi.

```
go run *.go
```

Jangan menggunakan `go run main.go`, dikarenakan dalam package `main` terdapat beberapa file lain yang harus diikut-sertakan pada saat runtime.

```
[inovalagung:chapter-1.18 $ go run *.go
server started at localhost:9000]
```

Test web service kecil ini menggunakan command `curl`.

```
$ curl -X GET --user batman:secret http://localhost:9000/student
```

```
$ curl -X GET --user batman:secret http://localhost:9000/student?id=s001
```

```
[novalagung:chapter-1.18 $ curl -X GET --user batman:secret http://localhost:9000/student?id=s001 ]  
{"Id":"s001","Name":"bourne","Grade":2}  
[novalagung:chapter-1.18 $ curl -X GET --user batman:secret http://localhost:9000/student ]  
[{"Id":"s001","Name":"bourne","Grade":2}, {"Id":"s002","Name":"ethan","Grade":2}, {"Id":"s003","Name":  
": "wick", "Grade":3}]  
novalagung:chapter-1.18 $ ]
```

## B.19. Middleware http.Handler

Pada bab ini, kita akan belajar penggunaan interface `http.Handler` untuk implementasi custom middleware. Kita akan menggunakan sample proyek pada bab sebelumnya [A18 - HTTP Basic Auth](#) sebagai dasar bahan pembahasan bab ini.

Apa itu middleware? Istilah middleware berbeda-beda di tiap bahasa/framework. NodeJS dan Rails ada istilah middleware. Pada pemrograman Java Enterprise, istilah filters digunakan. Pada C# istilahnya adalah delegate handlers. Definisi dari middleware sendiri versi penulis, sebuah blok kode yang dipanggil sebelum ataupun sesudah http request di proses.

Di bab sebelumnya, kalau dilihat, ada beberapa proses yang dijalankan dalam handler route `/student`, yaitu pengecekan otentikasi dan pengecekan method. Misalnya terdapat rute lagi, maka dua validasi tersebut juga harus dipanggil lagi dalam handlernya.

```
func ActionStudent(w http.ResponseWriter, r *http.Request) {
    if !Auth(w, r) { return }
    if !AllowOnlyGET(w, r) { return }

    // ...
}
```

Jika ada banyak rute, apa yang harus kita lakukan? salah satu solusi yang bisa digunakan adalah dengan memanggil fungsi `Auth()` dan `AllowOnlyGet()` di semua handler rute yang ada. Namun jelasnya ini bukan best practice. Dan juga belum tentu di tiap rute hanya ada dua validasi ini, bisa saja ada lebih banyak proses, misalnya pengecekan csrf, authorization, dan lainnya.

Solusi dari masalah tersebut adalah, mengkonversi fungsi-fungsi di atas menjadi middleware.

### B.19.1. Interface http.Handler

Interface `http.Handler` merupakan tipe data paling populer di golang untuk keperluan manajemen middleware. Struct yang mengimplementasikan interface ini diwajibkan memiliki method dengan skema `ServeHTTP(ResponseWriter, *Request)`.

Di golang sendiri objek utama untuk keperluan routing yaitu `mux` atau multiplexer, adalah mengimplementasikan interface `http.Handler` ini.

Dengan memanfaatkan interface ini, kita akan membuat beberapa middleware. Fungsi pengecekan otentikasi dan pengecekan method akan kita ubah menjadi middleware terpisah.

### B.19.2. Persiapan

OK, mari kita praktikan. Pertama duplikat folder projek sebelumnya sebagai folder proyek baru. Lalu pada `main.go`, ubah isi fungsi `ActionStudent` dan `main`.

- Fungsi `ActionStudent()`

```
func ActionStudent(w http.ResponseWriter, r *http.Request) {
    if id := r.URL.Query().Get("id"); id != "" {
        OutputJSON(w, SelectStudent(id))
        return
    }
}
```

```
    OutputJSON(w, GetStudents())
}
```

- Fungsi `main()`

```
func main() {
    mux := http.DefaultServeMux

    mux.HandleFunc("/student", ActionStudent)

    var handler http.Handler = mux
    handler = MiddlewareAuth(handler)
    handler = MiddlewareAllowOnlyGet(handler)

    server := new(http.Server)
    server.Addr = ":9000"
    server.Handler = handler

    fmt.Println("server started at localhost:9000")
    server.ListenAndServe()
}
```

Perubahan pada kode `ActionStudent()` adalah, pengecekan basic auth dan pengecekan method dihapus. Selain itu di fungsi `main()` juga terdapat cukup banyak perubahan, yang detailnya akan kita bahas di bawah ini.

### B.19.3. Mux / Multiplexer

Di golang, mux (kependekan dari multiplexer) adalah router. Semua routing pasti dilakukan lewat objek mux ini.

Apa benar? routing `http.HandleFunc()` sepertinya tidak menggunakan mux? Sebenarnya routing tersebut juga menggunakan mux. Golang memiliki default objek mux yaitu `http.DefaultServeMux`. Routing yang langsung dilakukan dari fungsi `HandleFunc()` milik package `net/http` sebenarnya mengarah ke method default mux `http.DefaultServeMux.HandleFunc()`. Jadi kedua gaya routing berikut hasilnya adalah sama. Silakan lihat kode berikut untuk perbandingan.

```
http.HandleFunc("/student", ActionStudent)

// vs

mux := http.DefaultServeMux
mux.HandleFunc("/student", ActionStudent)
```

Mux sendiri adalah bentuk nyata struct yang mengimplementasikan interface `http.Handler`. Untuk lebih jelasnya silakan baca dokumentasi package net/http di <https://golang.org/pkg/net/http/#Handle>.

Kembali ke pembahasan source code. Di kode setelah routing, bisa dilihat objek `mux` ditampung ke variabel baru bertipe `http.Handler`. Seperti ini adalah valid karena memang struct multiplexer memenuhi kriteria interface `http.Handler`, yaitu memiliki method `ServeHTTP()`.

Lalu dari objek `handler` tersebut, ke-dua middleware dipanggil dengan parameter adalah objek `handler` itu sendiri dan nilai baliknya ditampung pada objek yang sama.

```
var handler http.Handler = mux
handler = MiddlewareAuth(handler)
handler = MiddlewareAllowOnlyGet(handler)
```

Fungsi `MiddlewareAuth()` dan `MiddlewareAllowOnlyGet()` adalah middleware yang akan kita buat setelah ini. Cara registrasi middleware yang paling populer adalah dengan memanggilnya secara sekuensial atau berurutan, seperti pada kode di atas.

- `MiddlewareAuth()` bertugas untuk melakukan pengecekan credentials, basic auth.
- `MiddlewareAllowOnlyGet()` bertugas untuk melakukan pengecekan method.

Silakan lihat source code beberapa library middleware yang sudah terkenal seperti gorilla, gin-contrib, echo middleware, dan lainnya; kesemuanya metode implementasi middleware-nya adalah sama, atau paling tidak mirip. Point plus nya, beberapa diantara library tersebut mudah diintegrasikan dan compatible satu sama lain.

Kedua middleware yang akan kita buat tersebut mengembalikan fungsi bertipe `http.Handler`. Eksekusi middleware sendiri terjadi pada saat ada http request masuk.

Setelah semua middleware diregistrasi. Masukan objek `handler` ke property `.Handler` milik server.

```
server := new(http.Server)
server.Addr = ":9000"
server.Handler = handler
```

### B.19.3. Pembuatan Middleware

Di dalam `middleware.go` ubah fungsi `Auth()` (hasil salinan projek pada bab sebelumnya) menjadi fungsi `MiddlewareAuth()`. Parameternya objek bertipe `http.Handler`, dan nilai baliknya juga sama.

```
func MiddlewareAuth(next http.Handler) http.Handler {
    return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
        username, password, ok := r.BasicAuth()
        if !ok {
            w.Write([]byte(`something went wrong`))
            return
        }

        isValid := (username == USERNAME) && (password == PASSWORD)
        if !isValid {
            w.Write([]byte(`wrong username/password`))
            return
        }

        next.ServeHTTP(w, r)
    })
}
```

Idealnya fungsi middleware harus mengembalikan struct yang implements `http.Handler`. Beruntungnya, goLang sudah menyiapkan fungsi ajaib untuk mempersingkat pembuatan struct-yang-implements- `http.Handler`. Fungsi tersebut adalah `http.HandlerFunc`, cukup bungkus callback `func(http.ResponseWriter, *http.Request)` sebagai tipe `http.HandlerFunc` dan semuanya beres.

Isi dari `MiddlewareAuth()` sendiri adalah pengecekan basic auth, sama seperti pada bab sebelumnya.

Tak lupa, ubah juga `AllowOnlyGet()` menjadi `MiddlewareAllowOnlyGet()`.

```
func MiddlewareAllowOnlyGet(next http.Handler) http.Handler {
    return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
        if r.Method != "GET" {
            w.Write([]byte("Only GET is allowed"))
            return
        }
    })
}
```

```
    next.ServeHTTP(w, r)
})
}
```

## B.19.4. Testing

Jalankan aplikasi.

```
[novalagung:chapter-1.18 $ go run *.go
server started at localhost:9000
```

Lalu test menggunakan `curl`, hasilnya adalah sama dengan pada bab sebelumnya.

```
[novalagung:chapter-1.18 $ curl -X GET --user batman:secret http://localhost:9000/student?id=s001 ]
{"Id":"s001","Name":"bourne","Grade":2}
[novalagung:chapter-1.18 $ curl -X GET --user batman:secret http://localhost:9000/student
[{"Id":"s001","Name":"bourne","Grade":2}, {"Id":"s002","Name":"ethan","Grade":2}, {"Id":"s003","Name
":"wick","Grade":3}]
novalagung:chapter-1.18 $ ]
```

Dibanding metode pada bab sebelumnya, dengan teknik ini kita bisa sangat mudah mengontrol lalu lintas routing aplikasi, karena semua rute pasti melewati middleware terlebih dahulu sebelum sampai ke tujuan. Cukup maksimalkan middleware tersebut tanpa mengganggu fungsi callback masing-masing rute.

## B.20. Custom Multiplexer

Pada bab ini, kita akan belajar membuat custom multiplexer, memanfaatkannya untuk mempermudah manajemen middleware.

Silakan salin projek sebelumnya, [bab A19 - Middleware http.Handler](#), ke folder baru untuk keperluan pembelajaran.

### B.20.1. Pembuatan Custom Mux

Pada bab sebelumnya, default mux milik golang digunakan untuk routing dan implementasi middleware. Kali ini default mux tersebut tidak digunakan, mux baru akan dibuat.

Namun pembuatan mux baru tidaklah cukup, karena fungsinya tidak akan ada bedanya dibanding default mux. Agar lebih berguna, kita akan buat tipe mux baru, meng-embed `http.ServeMux` kedalamnya, lalu membuat beberapa hal dalam struct tersebut.

OK, langsung saja kita praktikan. Ubah isi fungsi main menjadi seperti berikut.

```
mux := new(CustomMux)

mux.HandleFunc("/student", ActionStudent)

mux.RegisterMiddleware(MiddlewareAuth)
mux.RegisterMiddleware(MiddlewareAllowOnlyGet)

server := new(http.Server)
server.Addr = ":9000"
server.Handler = mux

fmt.Println("server started at localhost:9000")
server.ListenAndServe()
```

Objek `mux` dicetak dari struct `CustomMux` yang jelasnya akan dibuat. Struct ini di dalamnya meng-embed `http.ServeMux`.

Registrasi middleware juga diubah, sekarang menggunakan method `.RegisterMiddleware()` milik mux.

Pada file `middleware.go`, siapkan struct `CustomMux`. Selain meng-embed objek mux milik golang, siapkan juga satu variabel bertipe slice-dari-tipe-fungsi-middleware.

```
type CustomMux struct {
    http.ServeMux
    middlewares []func(next http.Handler) http.Handler
}
```

Buat fungsi `RegisterMiddleware()`. Middleware yang didaftarkan ditampung oleh slice `.middlewares`.

```
func (c *CustomMux) RegisterMiddleware(next func(next http.Handler) http.Handler) {
    c.middlewares = append(c.middlewares, next)
}
```

Lalu buat method `ServeHTTP`. Method ini diperlukan dalam custom mux agar memenuhi kriteria interface `http.Handler`.

```
func (c *CustomMux) ServeHTTP(w http.ResponseWriter, r *http.Request) {
```

```

var current http.Handler = &c.ServeMux

for _, next := range c.middlewares {
    current = next(current)
}

current.ServeHTTP(w, r)
}

```

Method `ServeHTTP()` milik mux adalah method yang pasti dipanggil pada web server, di setiap request yang masuk.

Dengan perubahan di atas, setiap kali ada request masuk pasti akan melewati middleware-middleware terlebih dahulu secara berurutan. Jika lolos middleware ke-1, lanjut ke-2; jika lolos middleware ke-2, lanjut ke-3; dan seterusnya.

## B.20.2. Testing

Jalankan aplikasi.

```
[novalagung:chapter-1.18 $ go run *.go
server started at localhost:9000
```

Lalu test menggunakan `curl`, hasilnya adalah sama dengan pada bab sebelumnya.

```
[novalagung:chapter-1.18 $ curl -X GET --user batman:secret http://localhost:9000/student?id=s001 ]
{"Id":"s001","Name":"bourne","Grade":2}
[novalagung:chapter-1.18 $ curl -X GET --user batman:secret http://localhost:9000/student ]
[{"Id":"s001","Name":"bourne","Grade":2}, {"Id":"s002","Name":"ethan","Grade":2}, {"Id":"s003","Name":
"wick","Grade":3}]
novalagung:chapter-1.18 $
```

Jika ada keperluan untuk menambahkan middleware baru lainnya, cukup registrasikan lewat `.RegisterMiddleware()`. Source code menjadi lebih rapi dan nyaman untuk dilihat.

## B.21. HTTP Cookie

Cookie adalah data dalam bentuk teks yang disimpan pada komputer (oleh web browser) ketika pengunjung sedang surfing ke sebuah situs. Cookie dapat dibuat dari sisi front end (javascript) maupun back end (dalam konteks ini golang).

Cookie merupakan salah satu aspek penting dalam pengembangan aplikasi web. Sangat sering kita membutuhkan sebuah data bisa disimpan dan diakses untuk keperluan aplikasi web kita, seperti pengecekan preferensi pengunjung, pengecekan status login tidaknya user.

Pada bab ini kita akan belajar bagaimana cara membuat dan mengakses cookie di golang.

### B.21.1. Praktek

Buat sebuah folder proyek, siapkan satu buah file `main.go`. Buat fungsi `main()`, registrasikan dua buah rute.

```
package main

import (
    "fmt"
    "github.com/novalagung/gubrak"
    "net/http"
    "time"
)

type M map[string]interface{}

var cookieName = "CookieData"

func main() {
    http.HandleFunc("/", ActionIndex)
    http.HandleFunc("/delete", ActionDelete)

    fmt.Println("server started at localhost:9000")
    http.ListenAndServe(":9000", nil)
}
```

Variabel `cookieName` berisikan string, akan kita gunakan sebagai nama cookie.

- Rute `/` bertugas untuk membuat cookie baru (jika belum ada atau cookie sudah ada namun expired).
- Rute `/delete` mempunyai tugas untuk menghapus cookie, lalu redirect ke `/` sehingga cookie baru akan dibuat

OK, sekarang buat fungsi handler `ActionIndex()`. Di dalam fungsi ini, data berupa random string disimpan dalam cookie.

```
func ActionIndex(w http.ResponseWriter, r *http.Request) {
    cookieName := "CookieData"

    c := &http.Cookie{}

    if storedCookie, _ := r.Cookie(cookieName); storedCookie != nil {
        c = storedCookie
    }

    if c.Value == "" {
        c = &http.Cookie{}
        c.Name = cookieName
        c.Value = gubrak.RandomString(32)
```

```

        c.Expires = time.Now().Add(5 * time.Minute)
        http.SetCookie(w, c)
    }

    w.Write([]byte(c.Value))
}

```

Cookie bisa dikases lewat method `.Cookie()` milik objek `*http.Request`. Method ini mengembalikan 2 informasi:

- Objek cookie
- Error, jika ada

Pada kode di atas, ketika `storedCookie` nilainya bukanlah `nil` (berarti cookie dengan nama `cookieName` sudah dibuat), maka objek cookie tersebut disimpan dalam `c`.

Pembuatan cookie cukup mudah, tinggal cetak saja objek baru dari struct `http.Cookie`.

Jika `c.value` adalah kosong, kita asumsikan bahwa cookie belum pernah dibuat (atau expired), maka kita buat cookie baru dengan data adalah random string.

Untuk mempermudah generate random string, kita gunakan library bernama [gubrak](#). Fungsi `gubrak.RandomString(32)` akan menghasilkan string ajak 32 karakter.

Cookie bisa expired. Lama cookie aktif ditentukan lewat property `Expires`. Pada kode di atas expiration duration kita set selama 5 menit.

Gunakan `http.SetCookie()` untuk menyimpan cookie yang baru dibuat.

OK, selanjutnya buat handler `ActionDelete()`, seperti yang sudah disinggung di atas. Handler ini difungsikan untuk menghapus cookie dengan nama `cookieName`, lalu redirect ke `/` agar cookie baru diciptakan.

```

func ActionDelete(w http.ResponseWriter, r *http.Request) {
    c := &http.Cookie{}
    c.Name = cookieName
    c.Expires = time.Unix(0, 0)
    c.MaxAge = -1
    http.SetCookie(w, c)

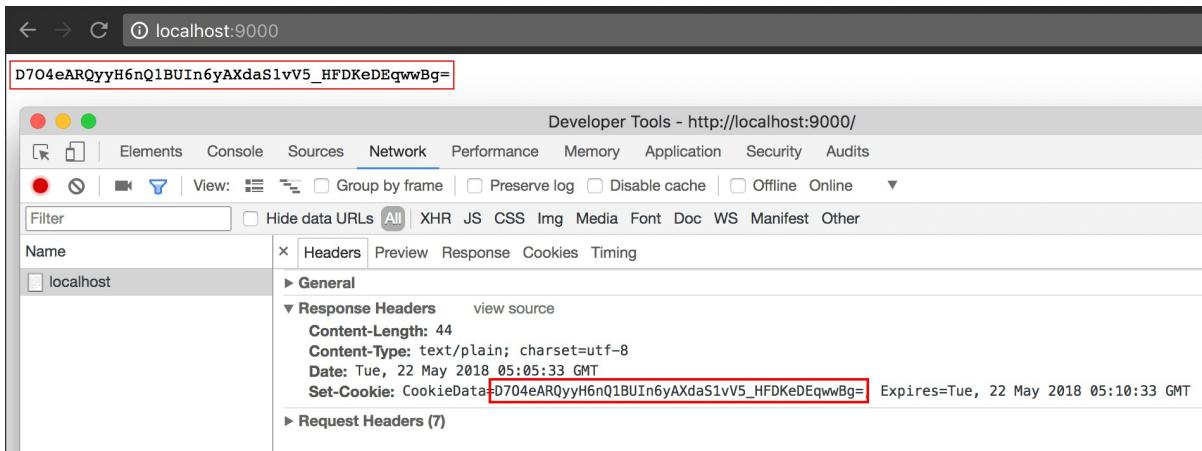
    http.Redirect(w, r, "/", http.StatusTemporaryRedirect)
}

```

Cara menghapus cookie adalah dengan menge-set ulang cookie dengan nama yang sama, dengan isi property `Expires = time.Unix(0, 0)` dan `MaxAge = -1`. Tujuannya agar cookie expired.

## B.21.2. Testing

Jalankan aplikasi, lalu akses `/`. Sebuah random string akan muncul di layar, dan jika kita cek pada bagian response header, informasi cookie nya juga tampil.



Coba refresh page beberapa kali, informasi header cookie dan data yang muncul adalah tetap sama. Karena ketika cookie sudah pernah dibuat, maka seterusnya endpoint ini akan menggunakan data cookie yang sudah tersimpan tersebut.

Selanjutnya, buka url `/delete`, halaman akan di redirect kembali ke `/`, dan random string baru beserta cookie baru terbuat. Dalam endpoint ini, cookie dihapus, dan karena step selanjutnya adalah redirect ke `/`, maka proses pengecekan dan pembuatan cookie akan dimulai kembali. Pengunjung akan mendapatkan data cookie baru dengan nama yang sama.

### B.21.3. Properties Object `http.Cookie`

Objek cookie memiliki beberapa property, beberapa diantaranya:

Property	Tipe Data	Deskripsi
Value	string	Data yang disimpan di cookie
Path	string	Scope path cookie
Domain	string	Scope domain cookie
Expires	time.Time	Durasi cookie, ditulis dalam tipe <code>time.Time</code>
MaxAge	int	Durasi cookie, ditulis dalam detik (numerik)
Secure	bool	Scope cookie dalam konteks protocol yang digunakan ketika pengaksesan web. Properti ini hanya berguna pada saat web server SSL/TLS enabled. <ul style="list-style-type: none"> <li>Jika <code>false</code>, maka cookie yang disimpan ketika web diakses menggunakan protokol <code>http://</code>, tetapi bisa diakses lewat <code>https://</code>, dan berlaku juga untuk kebalikannya.</li> <li>Jika <code>true</code>, pada saat pengaksesan lewat protokol <code>https://</code>, maka data cookie akan di-enkripsi. Sedangkan pada pengaksesan lewat protokol <code>http://</code> cookie disimpan seperti biasa (tanpa dienkripsi). Jika dalam satu web server, dua protokol tersebut bisa diakses, <code>https://</code> dan <code>http://</code>, maka aturan di atas tetap berlaku untuk masing-masing protokol, dengan catatan data yang disimpan lewat <code>https://</code> hanya bisa diakses lewat protokol tersebut.</li> </ul>
HttpOnly	bool	<ul style="list-style-type: none"> <li>Jika <code>false</code>, maka cookie bisa dibuat lewat back end (golang), maupun lewat front end (javascript)</li> <li>Jika <code>true</code>, maka cookie hanya bisa diciptakan dari back end</li> </ul>

- [Gubrak](#), by Noval Agung, MIT license



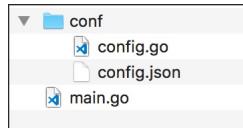
## B.22. Configuration File

Dalam development, pasti banyak sekali variabel dan konstanta yang diperlukan. Mulai dari variabel yang dibutuhkan untuk start server seperti port, timeout, hingga variabel global dan variabel shared lainnya.

Pada bab ini, kita akan belajar cara membuat config file modular.

### B.22.1. Struktur Aplikasi

Pertama-tama, buat project baru, siapkan dengan struktur seperti gambar berikut.



Folder `conf` berisi 2 file.

1. File `config.json`. Semua konfigurasi nantinya harus disimpan di file ini dalam struktur JSON.
2. File `config.go`. Berisikan beberapa fungsi dan operasi untuk mempermudah pengaksesan konfigurasi dari file `config.json`.

### B.22.2. File Konfigurasi JSON `config.json`

Semua konfigurasi perlu dituliskan dalam file ini. Desain struktur JSON nya untuk bisa mudah dipahami. Tulis data berikut di file tersebut.

```
{
  "server": {
    "port": 9000,
    "read_timeout": 5,
    "write_timeout": 5
  },
  "log": {
    "verbose": true
  }
}
```

Ada 4 buah konfigurasi disiapkan.

1. Property `server.port`. Port yang digunakan saat start web server.
2. Property `server.read_timeout`. Dijadikan sebagai timeout read.
3. Property `server.write_timeout`. Dijadikan sebagai timeout write.
4. Property `log.verbose`. Penentu apakah log di print atau tidak.

### B.22.3. Pemrosesan Konfigurasi

Pada file `config.go`, nantinya kita akan buat sebuah fungsi, isinya mengembalikan objek cetakan struct representasi dari `config.json`.

Siapkan struct nya terlebih dahulu.

```

package conf

import (
    "encoding/json"
    "io/ioutil"
    "os"
    "path/filepath"
    "time"
)

var shared *_Configuration

type _Configuration struct {
    Server struct {
        Port           int      `json:"port"`
        ReadTimeout   time.Duration `json:"read_timeout"`
        WriteTimeout  time.Duration `json:"write_timeout"`
    } `json:"server"`

    Log struct {
        Verbose bool `json:"verbose"`
    } `json:"log"`
}

```

Bisa dilihat pada kode di atas, struct bernama `_Configuration` dibuat. Struct ini berisikan banyak property yang strukturnya sama persis dengan isi file `config.json`. Dengan desain seperti ini, akan sangat memudahkan developer dalam pengaksesan konfigurasi.

Dari struct tersebut tercetak private objek bernama `shared`. Variabel inilah yang nantinya akan dikembalikan lewat fungsi yang akan kita buat.

Selanjutnya, isi `init()` dengan beberapa proses: membaca file json, lalu di decode ke object `shared`.

Dengan menuliskan proses barusan ke fungsi `init()`, pada saat package `conf` ini di import ke package lain maka file `config.json` akan otomatis di parsing. Dan dengan menambahkan sedikit validasi, parsing hanya akan terjadi sekali di awal.

```

func init() {
    if shared != nil {
        return
    }

    basePath, err := os.Getwd()
    if err != nil {
        panic(err)
        return
    }

    bts, err := ioutil.ReadFile(filepath.Join(basePath, "conf", "config.json"))
    if err != nil {
        panic(err)
        return
    }

    shared = new(_Configuration)
    err = json.Unmarshal(bts, &shared)
    if err != nil {
        panic(err)
        return
    }
}

```

Lalu buat fungsi yang mengembalikan object `shared`.

```
func Configuration() _Configuration {
    return *shared
}
```

## B.22.4. Routing & Server

Masuk ke bagian implementasi, buka `main.go`, lalu buat cusom mux.

```
package main

import (
    "chapter-B.22/conf"
    "fmt"
    "log"
    "net/http"
    "time"
)

type CustomMux struct {
    http.ServeMux
}

func (c CustomMux) ServeHTTP(w http.ResponseWriter, r *http.Request) {
    if conf.Configuration().Log.Verbose {
        log.Println("Incoming request from", r.Host, "accessing", r.URL.String())
    }

    c.ServeMux.ServeHTTP(w, r)
}
```

Bisa dilihat dalam method `ServeHTTP()` di atas, ada pengecekan salah satu konfigurasi, yaitu `Log.Verbose`. Cara pengaksesannya cukup mudah, yaitu lewat fungsi `Configuration()` milik package `conf` yang telah di-import.

OK, kembali lagi ke contoh, dari mux di atas, buat object baru bernama `router`, lalu lakukan registrasi beberapa rute.

```
func main() {
    router := new(CustomMux)
    router.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
        w.Write([]byte("Hello World!"))
    })
    router.HandleFunc("/howareyou", func(w http.ResponseWriter, r *http.Request) {
        w.Write([]byte("How are you?"))
    })

    // ...
}
```

Selanjutnya, kita akan start web server untuk serve mux di atas. Masih di dalam `main.go`, tambahkan kode berikut.

```
server := new(http.Server)
server.Handler = router
server.ReadTimeout = conf.Configuration().Server.ReadTimeout * time.Second
server.WriteTimeout = conf.Configuration().Server.WriteTimeout * time.Second
server.Addr = fmt.Sprintf(":%d", conf.Configuration().Server.Port)

if conf.Configuration().Log.Verbose {
    log.Printf("Starting server at %s \n", server.Addr)
}

err := server.ListenAndServe()
if err != nil {
    panic(err)
```

```
}
```

Objek baru bernama `server` telah dibuat dari struct `http.Server`. Untuk start server cukup panggil method `ListenAndServe()` milik objek tersebut.

Dengan memanfaatkan struct ini, kita bisa meng-custom beberapa konfigurasi default pada golang web server. Di antaranya seperti `ReadTimeout` dan `WriteTimeout`.

Pada kode di atas bisa kita lihat, ada 4 buah properti milik `server` di-isi.

- `server.Handler` . Properti ini wajib di isi dengan custom mux yang dibuat.
- `server.ReadTimeout` . Adalah timeout ketika memproses sebuah request. Kita isi dengan nilai dari configurasi.
- `server.WriteTimeout` . Adalah timeout ketika memproses response.
- `server.Addr` . Port yang digunakan web server pada saat start.

Terakhir jalankan aplikasi, akses dua buah endpoint yang sudah dibuat, lalu coba cek di console.

```
[novalagung:chapter-A.22 $ go run main.go
2018/06/04 07:54:27 Starting server at :9000
2018/06/04 07:54:28 Incoming request from localhost:9000 accessing /
2018/06/04 07:54:32 Incoming request from localhost:9000 accessing /howareyou
```

Coba ubah konfigurasi pada `config.json` nilai `log.verbose` menjadi `false` . Lalu restart aplikasi, maka log tidak muncul.

## C.1. Echo Framework & Routing

Pada bab ini kita akan belajar cara mudah routing menggunakan [Echo Framework](#).

Mulai bab B1 hingga B6 kita akan mempelajari banyak aspek dalam framework Echo dan mengkobinasikannya dengan beberapa library lain.

### C.1.1 Echo Framework

Echo adalah framework bahasa go yang cocok untuk pengembangan aplikasi web. Framework ini cukup terkenal di komunitas. Echo merupakan framework besar, didalamnya terdapat banyak sekali dependensi.

Salah satu dependensi yang ada didalamnya adalah router, dan pada bab ini kita akan mempelajarinya.

Dari banyak routing library yang sudah penulis gunakan, hampir kesemuanya mempunyai kemiripan dalam hal penggunaannya, cukup panggil fungsi/method yang dipilih (biasanya namanya sama dengan HTTP Method), lalu sisipkan rute pada parameter pertama dan handler pada parameter kedua.

Berikut contoh sederhana penggunaan echo framework.

```
r := echo.New()
r.GET("/", handler)
r.Start(":9000")
```

Sebuah objek router `r` dicetak lewat `echo.New()`. Lalu lewat objek router tersebut, dilakukan registrasi rute untuk `/` dengan method GET dan handler adalah closure `handler`. Terakhir, dari objek router di-startlah sebuah web server pada port 9000.

Echo router mengadopsi konsep [radix tree](#), membuat performa lookup nya begitu cepat. Tak juga itu, pemanfaatan sync pool membuat penggunaan memory lebih hemat, dan aman dari GC overhead.

### C.1.2. Praktek

Mari kita pelajari lebih lanjut dengan praktik langsung. Buat folder proyek baru, buat `main.go`, isi dengan kode berikut, kemudian jalankan aplikasi.

```
package main

import (
    "fmt"
    "github.com/labstack/echo"
    "net/http"
    "strings"
)

type M map[string]interface{}

func main() {
    r := echo.New()

    r.GET("/", func(ctx echo.Context) error {
        data := "Hello from /index"
        return ctx.String(http.StatusOK, data)
    })
}
```

```
r.Start(":9000")
}
```

Kode di atas adalah contoh sederhana penerapan echo router.



Routing dengan memanfaatkan package `net/http` dalam penerapannya adalah menggunakan `http.HandleFunc()` atau `http.Handle()`. Berbeda dengan Echo, routingnya adalah method-based, tidak hanya endpoint dan handler yang di-registrasi, method juga.

Statement `echo.New()` mengembalikan objek mux/router. Pada kode di atas rute `/` dengan method `GET` di-daftarkan. Selain `r.GET()` ada banyak lagi method lainnya, semua method dalam [spesifikasi REST](#) seperti `PUT`, `POST`, dan lainnya bisa digunakan.

Handler dari method routing milik echo membutuhkan satu argument saja, dengan tipe adalah `echo.Context`. Dari argumen tersebut objek `http.ResponseWriter` dan `http.Request` bisa di-akses. Namun kedua objek tersebut akan jarang kita gunakan karena `echo.Context` memiliki banyak method yang beberapa tugasnya sudah meng-cover operasi umum yang biasanya kita lakukan lewat objek request dan response, diantara seperti:

- Render output (dalam bentuk html, plain text, json, atau lainnya).
- Parsing request data (json payload, form data, query string).
- URL Redirection.
- ... dan lainnya.

Untuk mengakses objek `http.Request` gunakan `ctx.Request()`.

Sedang untuk objek `http.ResponseWriter` gunakan `ctx.Response()`.

Salah satu alasan lain kenapa penulis memilih framework ini, adalah karena desain route-handler-nya menarik. Dalam handler cukup kembalikan objek error ketika memang ada kesalahan terjadi, sedangkan jika tidak ada error maka kembalikan nilai `nil`.

Ketika terjadi error pada saat mengakses endpoint, idealnya [HTTP Status](#) error dikembalikan sesuai dengan jenis errornya. Tapi terkadang juga ada kebutuhan dalam kondisi tertentu `http.StatusOK` atau status 200 dikembalikan dengan disisipi informasi error dalam response body-nya. Kasus sejenis ini menjadikan standar error reporting menjadi kurang bagus. Pada konteks ini echo unggul menurut penulis, karena default-nya semua error dikembalikan sebagai response dalam bentuk yang sama.

Method `ctx.String()` dari objek context milik handler digunakan untuk mempermudah rendering data string sebagai output. Method ini mengembalikan objek error, jadi bisa digunakan langsung sebagai nilai balik handler. Argumen pertama adalah http status dan argumen ke-2 adalah data yang dijadikan output.

### C.1.3. Response Method milik `ctx`

Selain `ctx.String()` ada banyak method sejenis lainnya, berikut selengkapnya.

- **Method `.String()`**

Digunakan untuk render plain text sebagai output (isi response header `Content-Type` adalah `text/plain`). Method ini tugasnya sama dengan method `.write()` milik objek `http.ResponseWriter`.

```
r.GET("/index", func(ctx echo.Context) error {
    data := "Hello from /index"
    return ctx.String(http.StatusOK, data)
}
```

```
})
```

### • Method `.HTML()`

Digunakan untuk render html sebagai output. Isi response header `Content-Type` adalah `text/html`.

```
r.GET("/html", func(ctx echo.Context) error {
    data := "Hello from /html"
    return ctx.HTML(http.StatusOK, data)
})
```

### • Method `.Redirect()`

Digunakan untuk redirect, pengganti `http.Redirect()`.

```
r.GET("/index", func(ctx echo.Context) error {
    return ctx.Redirect(http.StatusTemporaryRedirect, "/")
})
```

### • Method `.JSON()`

Digunakan untuk render data JSON sebagai output. Isi response header `Content-Type` adalah `application/json`.

```
r.GET("/json", func(ctx echo.Context) error {
    data := M{"Message": "Hello", "Counter": 2}
    return ctx.JSON(http.StatusOK, data)
})
```

## C.1.4. Parsing Request

Echo juga menyediakan beberapa method untuk keperluan parsing request, diantaranya:

### • Parsing Query String

Method `.QueryParam()` digunakan untuk mengambil data pada query string request, sesuai dengan key yang diinginkan.

```
r.GET("/page1", func(ctx echo.Context) error {
    name := ctx.QueryParam("name")
    data := fmt.Sprintf("Hello %s", name)

    return ctx.String(http.StatusOK, data)
})
```

Test menggunakan curl:

```
curl -X GET http://localhost:9000/page1?name=grayson
```

### • Parsing URL Path Param

Method `.Param()` digunakan untuk mengambil data path parameter sesuai skema rute.

```
r.GET("/page2/:name", func(ctx echo.Context) error {
```

```

name := ctx.Param("name")
data := fmt.Sprintf("Hello %s", name)

return ctx.String(http.StatusOK, data)
})

```

Bisa dilihat, terdapat `:name` pada pendeklarasian rute. Nantinya url apapun yang ditulis sesuai skema di atas akan bisa diambil path parameter-nya. Misalkan `/page2/halo` maka `ctx.Param("name")` mengembalikan string `halo`.

Test menggunakan curl:

```
curl -X GET http://localhost:9000/page2/grayson
```

### • Parsing URL Path Param dan Setelahnya

Selain mengambil parameter sesuai spesifik path, kita juga bisa mengambil data **parameter path dan setelahnya**.

```

r.GET("/page3/:name/*", func(ctx echo.Context) error {
    name := ctx.Param("name")
    message := ctx.Param("*")

    data := fmt.Sprintf("Hello %s, I have message for you: %s", name, message)

    return ctx.String(http.StatusOK, data)
})

```

Statement `ctx.Param("*")` mengembalikan semua path sesuai dengan skema url-nya. Misal url adalah `/page3/tim/a/b/c/d/e/f/g/h` maka yang dikembalikan adalah `a/b/c/d/e/f/g/h`.

Test menggunakan curl:

```
curl -X GET http://localhost:9000/page3/tim/need/some/sleep
```

### • Parsing Form Data

Data yang dikirim sebagai request body dengan jenis adalah Form Data bisa di-ambil dengan mudah menggunakan `ctx.FormValue()`.

```

r.POST("/page4", func(ctx echo.Context) error {
    name := ctx.FormValue("name")
    message := ctx.FormValue("message")

    data := fmt.Sprintf(
        "Hello %s, I have message for you: %s",
        name,
        strings.Replace(message, "/", "", 1),
    )

    return ctx.String(http.StatusOK, data)
})

```

Test menggunakan curl:

```
curl -X POST -F name=damian -F message=angry http://localhost:9000/page4
```

Pada bab selanjutnya kita akan belajar teknik parsing request data yang lebih advance.

## C.1.5. Penggunaan `echo.WrapHandler` Untuk Routing Handler Bertipe `func(http.ResponseWriter, *http.Request)` atau `http.HandlerFunc`

Echo bisa dikombinasikan dengan handler ber-skema *NON-echo-handler* seperti

```
func(http.ResponseWriter, *http.Request) atau http.HandlerFunc .
```

Caranya dengan memanfaatkan fungsi `echo.WrapHandler` untuk mengkonversi handler tersebut menjadi echo-compatible. Lebih jelasnya silakan lihat kode berikut.

```
var ActionIndex = func(w http.ResponseWriter, r *http.Request) {
    w.Write([]byte("from action index"))
}

var ActionHome = http.HandlerFunc(
    func(w http.ResponseWriter, r *http.Request) {
        w.Write([]byte("from action home"))
    },
)

var ActionAbout = echo.WrapHandler(
    http.HandlerFunc(
        func(w http.ResponseWriter, r *http.Request) {
            w.Write([]byte("from action about"))
        },
    ),
)

func main() {
    r := echo.New()

    r.GET("/index", echo.WrapHandler(http.HandlerFunc(ActionIndex)))
    r.GET("/home", echo.WrapHandler(ActionHome))
    r.GET("/about", ActionAbout)

    r.Start(":9000")
}
```

Untuk routing handler dengan skema `func(http.ResponseWriter, *http.Request)`, maka harus dibungkus dua kali, pertama menggunakan `http.HandlerFunc`, lalu dengan `echo.WrapHandler`.

Sedangkan untuk handler yang sudah bertipe `http.HandlerFunc`, bungkus langsung menggunakan `echo.WrapHandler`.

## C.1.6. Routing Static Assets

Cara routing static assets di echo sangatlah mudah. Gunakan method `.Static()`, isi parameter pertama dengan prefix rute yang di-inginkan, dan parameter ke-2 dengan path folder tujuan.

Buat sub folder dengan nama `assets` dalam folder projek. Dalam folder tersebut buat sebuah file `layout.js`, isinya bebas.

Pada `main.go` tambahkan routing static yang mengarah ke path folder `assets`.

```
r.Static("/static", "assets")
```

Jalankan aplikasi, lalu coba akses `http://localhost:9000/static/layout.js`.



```
← → ⌂ ⓘ localhost:9000/static/layout.js
console.log('hello')
```

- 
- [Echo](#), by Vishal Rana (Lab Stack), MIT license

## C.2. Parsing HTTP Request Payload (Echo)

Pada bab ini kita akan belajar cara parsing beberapa variasi request payload.

Payload dalam HTTP request bisa dikirimkan dalam berbagai bentuk. Kita akan mempelajari cara untuk handle 4 jenis payload berikut.

- Form Data
- JSON Payload
- XML Payload
- Query String

Secara tidak sadar, kita sudah sering menggunakan jenis payload form data. Contohnya pada html form, ketika event submit di-trigger, data dikirim ke destinasi dalam bentuk form data. Indikatornya adalah data tersebut ditampung dalam request body, dan isi request header `Content-Type` adalah `application/x-www-form-urlencoded` (atau `multipart/form-data`).

JSON payload dan XML payload sebenarnya sama dengan Form Data, pembedanya adalah header `Content-Type` masing-masing request. Untuk JSON payload isi header tersebut adalah `application/json`, sedang untuk XML payload adalah `application/xml`.

Sedang jenis request query string adalah yang paling berbeda. Data tidak disisipkan dalam request body, melainkan pada url nya dalam bentuk key-value.

### C.2.1. Parsing Request Payload

Cara parsing payload request dalam echo sangat mudah, apapun jenis payload nya, API yang digunakan untuk parsing adalah sama.

Mari kita langsung praktekan, buat satu folder projek baru, buat `main.go`. Buat struct `User`, nantinya digunakan untuk menampung data payload yang dikirim.

```
package main

import (
    "fmt"
    "github.com/labstack/echo"
    "net/http"
)

type User struct {
    Name string `json:"name" form:"name" query:"name"`
    Email string `json:"email" form:"email" query:"email"`
}

func main() {
    r := echo.New()

    // routes here

    fmt.Println("server started at :9000")
    r.Start(":9000")
}
```

Selanjutnya siapkan satu buah endpoint `/user` menggunakan `r.Any()`. Method `.Any()` menerima segala jenis request dengan method GET, POST, PUT, atau lainnya.

```
r.Any("/user", func(c echo.Context) (err error) {
    u := new(User)
    if err = c.Bind(u); err != nil {
        return
    }

    return c.JSON(http.StatusOK, u)
})
```

Bisa dilihat dalam handler, method `.Bind()` milik `echo.Context` digunakan, dengan disisipi parameter pointer objek (hasil cetakan struct `User`). Parameter tersebut nantinya akan menampung payload yang dikirim, entah apapun jenisnya.

## C.2.2 Testing

Jalankan aplikasi, lakukan testing. Bisa gunakan `curl` ataupun API testing tools sejenis postman atau lainnya.

Di bawah ini shortcut untuk melakukan request menggunakan `curl` pada 4 jenis payload yang kita telah bahas. Response dari kesemua request adalah sama, menandakan bahwa data yang dikirim berhasil ditampung.

- **Form Data**

```
curl -X POST http://localhost:9000/user \
-d 'name=Joe' \
-d 'email=nope@novalagung.com'

# output => {"name":"Nope","email":"nope@novalagung.com"}
```

- **JSON Payload**

```
curl -X POST http://localhost:9000/user \
-H 'Content-Type: application/json' \
-d '{"name": "Nope", "email": "nope@novalagung.com"}'

# output => {"name": "Nope", "email": "nope@novalagung.com"}
```

- **XML Payload**

```
curl -X POST http://localhost:9000/user \
-H 'Content-Type: application/xml' \
-d '<?xml version="1.0"?>\n<Data>\n    <Name>Joe</Name>\n    <Email>nope@novalagung.com</Email>\n</Data>'

# output => {"name": "Nope", "email": "nope@novalagung.com"}
```

- **Query String**

```
curl -X GET http://localhost:9000/user?name=Joe&email=nope@novalagung.com

# output => {"name": "Nope", "email": "nope@novalagung.com"}
```

- [Echo](#), by Vishal Rana (Lab Stack), MIT license

## C.3. HTTP Request Payload Validation (Validator v9, Echo)

Pada bab ini kita akan belajar cara validasi payload request di sisi back end. Library yang kita gunakan adalah [gopkg.in/go-playground/validator.v9](https://gopkg.in/go-playground/validator.v9), library ini sangat berguna untuk keperluan validasi data.

### C.3.1. Payload Validation

Penggunaan validator cukup mudah, di struct penampung payload, tambahkan tag baru pada masing-masing property dengan skema `validate:"<rules>"`.

Langsung saja kita praktikan, buat folder projek baru dengan isi file `main.go`, lalu tulis kode berikut kedalamnya.

```
package main

import (
    "github.com/labstack/echo"
    "gopkg.in/go-playground/validator.v9"
    "net/http"
)

type User struct {
    Name  string `json:"name" validate:"required"`
    Email string `json:"email" validate:"required,email"`
    Age   int    `json:"age" validate:"gte=0,lte=80"`
}
```

Struct `User` memiliki 3 field, berisikan aturan/rule validasi, yang berbeda satu sama lain (bisa dilihat pada tag `validate`). Kita bahas validasi per-field agar lebih mudah untuk dipahami.

- Field `Name`, tidak boleh kosong.
- Field `Email`, tidak boleh kosong, dan isinya harus dalam format email.
- Field `Age`, tidak harus di-isi; namun jika ada isinya, maka harus berupa numerik dalam kisaran angka 0 hingga 80.

Kurang lebih berikut adalah penjelasan singkat mengenai beberapa rule yang kita gunakan di atas.

- Rule `required`, menandakan bahwa field harus di isi.
- Rule `email`, menandakan bahwa value pada field harus dalam bentuk email.
- Rule `gte=n`, artinya isi harus numerik dan harus di atas `n` atau sama dengan `n`.
- Rule `lte=n`, berarti isi juga harus numerik, dengan nilai di bawah `n` atau sama dengan `n`.

Jika sebuah field membutuhkan dua atau lebih rule, maka tulis kesemuanya dengan delimiter tanda koma ( , ).

OK, selanjutnya buat struct baru `CustomValidator` dengan isi sebuah property bertipe `*validator.Validate` dan satu buah method ber-skema `Validate(interface{})error`. Objek cetakan struct ini akan kita gunakan sebagai pengganti default validator milik echo.

```
type CustomValidator struct {
    validator *validator.Validate
}

func (cv *CustomValidator) Validate(i interface{}) error {
    return cv.validator.Struct(i)
}
```

```

func main() {
    e := echo.New()
    e.Validator = &CustomValidator{validator: validator.New()}

    // routes here

    e.Logger.Fatal(e.Start(":9000"))
}

```

Method `.Struct()` milik `*validator.Validate`, digunakan untuk mem-validasi data objek dari struct.

Library validator menyediakan banyak sekali cakupan data yang bisa divalidasi, tidak hanya struct, lebih jelasnya silakan lihat di laman github <https://github.com/go-playground/validator>.

Siapkan sebuah endpoint untuk keperluan testing. Dalam endpoint ini method `Validate` milik `CustomValidator` dipanggil.

```

e.POST("/users", func(c echo.Context) error {
    u := new(User)
    if err := c.Bind(u); err != nil {
        return err
    }
    if err := c.Validate(u); err != nil {
        return err
    }

    return c.JSON(http.StatusOK, true)
})

```

OK, jalankan aplikasi, lakukan testing.

Request	Status	Response Preview
1. Valid JSON (name: "nope", email: "nope@novalagung.com", age: 80)	200 OK	true
2. Invalid JSON (name: "nope", email: "novalagung.com", age: null)	200 OK	{"message": "Internal Server Error"}
3. Invalid JSON (name: "nope", email: "nope@novalagung.com", age: 120)	200 OK	{"message": "Internal Server Error"}
4. Invalid JSON (name: "nope", email: "nope@novalagung.com", age: null)	200 OK	true

Bisa dilihat pada gambar di atas, ada beberapa request yang mengembalikan error.

- Request pertama adalah valid.
- Request ke-2 error karena value dari field `email` tidak valid. Harusnya berisi value dalam format email.

- Request ke-3 error karena value field `age` lebih dari 80. Value seharusnya numerik kisaran 0 hingga 80.
- Sedangkan request ke-4 sukses meskipun `age` adalah `null`, hal ini karena rule untuk field tersebut tidak ada `required`.

Field `Age` tidak harus diisi; namun jika ada isinya, maka harus berupa numerik dalam kisaran angka 0 hingga 80.

Dari testing di atas bisa kita simpulkan bahwa fungsi validasi berjalan sesuai harapan. Namun masih ada yang kurang, ketika ada yang tidak valid, error yang dikembalikan selalu sama, yaitu message `Internal server error`.

Sebenarnya error 500 ini sudah sesuai jika muncul pada page yang sifatnya menampilkan konten. Pengguna tidak perlu tau secara mendetail mengenai detail error yang sedang terjadi. Mungkin dibuat saja halaman custom error agar lebih menarik.

Tapi untuk web service (RESTful API?), akan lebih baik jika errornya detail (terutama pada fase development), agar aplikasi consumer bisa lebih bagus dalam meng-handle error tersebut.

Nah, di bab selanjutnya kita akan belajar cara membuat custom error handler untuk meningkatkan kualitas error reporting.

---

- [Echo](#), by Vishal Rana (Lab Stack), MIT license
- [Validator v9](#), by Dean Karn (Go Playground), MIT license

## C.4. HTTP Error Handling (Validator v9, Echo)

Pada bab ini kita akan belajar cara membuat custom error handler yang lebih readable, sangat cocok untuk web service. Bahan dasar yang kita manfaatkan adalah source code pada bab sebelum ini, lalu dimodifikasi. Jadi silakan salin project pada bab sebelumnya sebagai projek folder baru.

### C.4.1. Error Handler

Cara meng-custom default error handler milik echo, adalah dengan meng-override property `e.HTTPErrorHandler`. Langsung saja override property tersebut dengan callback berisi parameter objek error dan context. Gunakan callback tersebut untuk bisa menampilkan error yg lebih detail.

```
e.HTTPErrorHandler = func(err error, c echo.Context) {
    report, ok := err.(*echo.HTTPError)
    if !ok {
        report = echo.NewHTTPError(http.StatusInternalServerError, err.Error())
    }

    c.Logger().Error(report)
    c.JSON(report.Code, report)
}
```

Pada kode di atas, objek `report` menampung objek error setelah di casting ke tipe `echo.HTTPError`. Error tipe ini adalah error-error yang berhubungan dengan http, yang di-handle oleh echo. Untuk error yang bukan dari echo, tipe nya adalah `error` biasa. Pada kode di atas kita standarkan, semua jenis error harus berbentuk `echo.HTTPError`.

Selanjutnya objek error tersebut kita tampilkan ke console dan juga ke browser dalam bentuk JSON.

OK, jalankan aplikasi, lalu test hasilnya.

```
[novalagung:chapter-b.4 $ go run 1-custom-api-error-simple.go]

      /_/_/_/_/
      /_/_/_/_\ v3.3.dev
      High performance, minimalist Go web framework
      https://echo.labstack.com
      _____0/
      0\

⇒ http server started on [::]:9000
{"time":"2018-07-13T15:45:45.383998212+07:00","level":"ERROR","prefix":"echo","file":"1-custom-api-error-simple.go","line":32,"message":"code=500, message=Key: 'User.Age' Error:Field validation for 'Age' failed on the 'lte' tag"}
```

### C.4.2. Human-Readable Error

Error yang dikembalikan sudah bisa lebih detail dibanding sebelumnya, tapi masih kurang, karena masih susah untuk dipahami. Lakukan modifikasi pada callback custom error handler menjadi seperti pada kode berikut.

```

e.HTTPErrorHandler = func(err error, c echo.Context) {
    report, ok := err.(*echo.HTTPError)
    if !ok {
        report = echo.NewHTTPError(http.StatusInternalServerError, err.Error())
    }

    if castedObject, ok := err.(validator.ValidationErrors); ok {
        for _, err := range castedObject {
            switch err.Tag() {
            case "required":
                report.Message = fmt.Sprintf("%s is required",
                    err.Field())
            case "email":
                report.Message = fmt.Sprintf("%s is not valid email",
                    err.Field())
            case "gte":
                report.Message = fmt.Sprintf("%s value must be greater than %s",
                    err.Field(), err.Param())
            case "lte":
                report.Message = fmt.Sprintf("%s value must be lower than %s",
                    err.Field(), err.Param())
            }
            break
        }
    }

    c.Logger().Error(report)
    c.JSON(report.Code, report)
}

```

Error yang dikembalikan oleh `validator.v9` bertipe `validator.ValidationErrors`. Pada kode di atas bisa kita lihat ada pengecekan apakah error tersebut adalah dari library `validator.v9` atau bukan; jika memang iya, maka `report.Message` diubah isinya dengan kata-kata yang lebih mudah dipahami.

Tipe `validator.ValidationErrors` sendiri sebenarnya merupakan slice `[]validator.FieldError`. Objek tersebut di-loop, lalu diambil-lah elemen pertama sebagai nilai bailk error.

OK, jalankan ulang aplikasi, lalu test.

POST ▾	http://localhost:9000/users	Send	<b>200 OK</b>	TIME 3.18 ms	SIZE 4 B	⌚ ▾
JSON ▾	Auth ▾	Query	Header ▾ 2	Docs	Preview ▾	Header ▾ 3
1 ▾ { 2     "name": "nope", 3     "email": "nope@novalagung.com", 4     "age": 80 5 }					1 true	Cookie
JSON ▾	Auth ▾	Query	Header ▾ 2	Docs	Preview ▾	Header ▾ 3
1 ▾ { 2     "name": "", 3     "email": "nope@novalagung.com", 4     "age": 80 5 }					1 { 2     "code": 500, 3     "message": "Name is required" 4 }	Cookie
JSON ▾	Auth ▾	Query	Header ▾ 2	Docs	Preview ▾	Header ▾ 3
1 ▾ { 2     "name": "nope", 3     "email": "novalagung.com", 4     "age": 80 5 }					1 { 2     "code": 500, 3     "message": "Email is not valid email" 4 }	Cookie
JSON ▾	Auth ▾	Query	Header ▾ 2	Docs	Preview ▾	Header ▾ 3
1 ▾ { 2     "name": "nope", 3     "email": "nope@novalagung.com", 4     "age": 120 5 }					1 { 2     "code": 500, 3     "message": "Age value must be lower than 80" 4 }	Cookie

### C.4.3. Custom Error Page

Untuk aplikasi non-web-service, akan lebih baik jika setiap terjadi error dimunculkan error page, atau halaman khusus yang menampilkan informasi error.

Di echo sangatlah mudah untuk membuat halaman custom error. Contoh implementasinya seperti kode di bawah ini.

```
e.HTTPErrorHandler = func(err error, c echo.Context) {
    report, ok := err.(*echo.HTTPError)
    if !ok {
        report = echo.NewHTTPError(http.StatusInternalServerError, err.Error())
    }

    errPage := fmt.Sprintf("%d.html", report.Code)
    if err := c.File(errPage); err != nil {
        c.HTML(report.Code, "Errrrrooooorrrrr")
    }
}
```

Setiap kali error terjadi, maka halaman dengan nama adalah `<error-code>.html` dicari untuk kemudian ditampilkan. Misalkan errornya adalah 500 Internal Server Error, maka halaman `500.html` muncul; error 404 Page Not Found, maka halaman `404.html` muncul.

Silakan ubah kode `fmt.Sprintf("%d.html", report.Code)` sesuai format nama halaman error yang digunakan.

- [Echo](#), by Vishal Rana (Lab Stack), MIT license
- [Validator v9](#), by Dean Karn (Go Playground), MIT license



## C.5. Template Rendering in Echo

Di bab ini kita akan belajar cara render template html pada aplikasi yang routingnya menggunakan echo.

Pada dasarnya proses parsing dan rendering template tidak di-handle oleh echo sendiri, melainkan oleh API dari package `html/template`. Jadi bisa dibilang cara render template di echo adalah sama seperti pada aplikasi yang murni menggunakan golang biasa, seperti yang sudah dibahas pada bab [B-4](#), [B-5](#), [B-9](#), dan [B-10](#).

Echo menyediakan satu fasilitas yang bisa kita manfaatkan untuk standarisasi rendering template. Cara penggunaannya, dengan meng-override default `.Renderer` property milik echo menggunakan objek cetakan struct, yang dimana pada struct tersebut harus ada method bernama `.Render()` dengan skema sesuai dengan kebutuhan echo. Nah, di dalam method `.Render()` inilah kode untuk parsing dan rendering template ditulis.

### C.5.1. Praktek

Agar lebih mudah dipahami, mari langsung kita praktikan. Siapkan sebuah projek, import package yang dibutuhkan.

```
package main

import (
    "github.com/labstack/echo"
    "html/template"
    "io"
    "net/http"
)

type M map[string]interface{}
```

Buat sebuah struct bernama `Renderer`, struct ini mempunyai 3 buah property dan 2 buah method.

```
type Renderer struct {
    template *template.Template
    debug    bool
    location string
}
```

Berikut adalah tugas dan penjelasan mengenai ketiga property di atas.

- Property `.template` bertanggung jawab untuk parsing dan rendering template.
- Property `.location` mengarah ke path folder dimana file template berada.
- Property `.debug` menampung nilai bertipe `bool`.
  - Jika `false`, maka parsing template hanya dilakukan sekali saja pada saat aplikasi di start. Mode ini sangat cocok untuk diaktifkan pada stage production.
  - Sedangkan jika nilai adalah `true`, maka parsing template dilakukan tiap pengaksesan rute. Mode ini cocok diaktifkan untuk stage development, karena perubahan kode pada file html sering pada stage ini.

Selanjutnya buat fungsi `NewRenderer()` untuk mempermudah inisialisasi objek renderer.

```
func NewRenderer(location string, debug bool) *Renderer {
    tpl := new(Renderer)
    tpl.location = location
    tpl.debug = debug

    tpl.ReloadTemplates()
```

```
    return tpl
}
```

Siapkan dua buah method untuk struct renderer, yaitu `.ReloadTemplates()` dan `.Render()`.

```
func (t *Renderer) ReloadTemplates() {
    t.template = template.Must(template.ParseGlob(t.location))
}

func (t *Renderer) Render(
    w io.Writer,
    name string,
    data interface{},
    c echo.Context,
) error {
    if t.debug {
        t.ReloadTemplates()
    }

    return t.template.ExecuteTemplate(w, name, data)
}
```

Method `.ReloadTemplates()` bertugas untuk parsing template. Method ini wajib dipanggil pada saat inisialisasi objek renderer. Jika `.debug == true`, maka method ini harus dipanggil setiap kali rute diakses (jika tidak, maka perubahan pada view tidak akan muncul).

Method `.Render()` berguna untuk render template yang sudah diparsing sebagai output. Method ini harus dibuat dalam skema berikut.

```
// skema method Render()
func (io.Writer, string, interface{}, echo.Context) error
```

Selanjutnya, buat echo router, override property renderer nya, dan siapkan sebuah rute.

```
func main() {
    e := echo.New()

    e.Renderer = NewRenderer("./*.html", true)

    e.GET("/index", func(c echo.Context) error {
        data := M{"message": "Hello World!"}
        return c.Render(http.StatusOK, "index.html", data)
    })

    e.Logger.Fatal(e.Start(":9000"))
}
```

Saat pemanggilan `NewRenderer()` sisipkan path folder tempat file template html berada. Gunakan `./*.html` agar mengarah ke **semua file html pada current folder**.

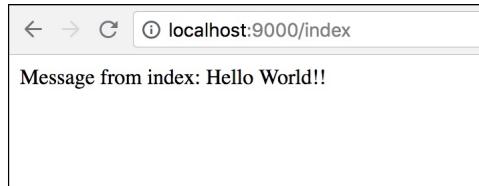
Buat file `index.html` dengan isi kode di bawah ini.

```
<!DOCTYPE html>
<html>
    <head>
        <title></title>
    </head>
    <body>
        Message from index: {{.message}}!
    </body>
</html>
```

Pada rute `/index`, sebuah variabel bernama `data` disiapkan, bertipe `map` dengan isi satu buah item. Data tersebut disisipkan pada saat view di-render, membuatnya bisa diakses dari dalam template html.

Syntax `{{ .message }}` artinya menampilkan isi property yang namanya adalah `message` dari current context (yaitu objek data yang disisipkan). Lebih jelasnya silakan baca kembali bab [B-6 - Template Actions & Variables](#).

Jalankan aplikasi untuk melihat hasilnya.



## C.5.2. Render Parsial dan Spesifik Template

Proses parsing dan rendering tidak di-handle oleh echo, melainkan menggunakan API dari `html/template`. Echo hanya menyediakan tempat untuk mempermudah pemanggilan fungsi rendernya. Nah dari sini berarti untuk render parsial, render spesifik template, maupun operasi template lainnya dilakukan seperti biasa, menggunakan `html/template`.

---

- [Echo](#), by Vishal Rana (Lab Stack), MIT license

## C.6. Advanced Middleware & Logging (Logrus, Echo Logger)

Middleware adalah sebuah blok kode yang dipanggil sebelum ataupun sesudah http request di-proses. Middleware biasanya dibuat per-fungsi-nya, contohnya: middleware autentikasi, middleware untuk logging, middleware untuk gzip compression, dan lainnya.

Pada bab ini kita akan belajar cara membuat dan me-manage middleware.

### C.6.1. Custom Middleware

Pembuatan middleware pada echo sangat mudah, cukup gunakan method `.use()` milik objek echo untuk registrasi middleware. Method ini bisa dipanggil berkali-kali, dan eksekusi middleware-nya sendiri adalah berurutan sesuai dengan urutan registrasi.

OK, langsung saja, buat folder projek baru dengan isi sebuah file `main.go` seperti biasanya. Lalu tulis kode berikut.

```
package main

import (
    "fmt"
    "github.com/labstack/echo"
    "net/http"
)

func main() {
    e := echo.New()

    // middleware here

    e.GET("/index", func(c echo.Context) (err error) {
        fmt.Println("threeeeeee!")

        return c.JSON(http.StatusOK, true)
    })

    e.Logger.Fatal(e.Start(":9000"))
}
```

Kode di atas merupakan aplikasi web kecil, berisi satu buah rute `/index` yang ketika di akses akan print log `"threeeeeee!"` ke console.

Selanjutnya, buat dua middleware, `middlewareOne` dan `middlewareTwo`. Isinya juga menampilkan log.

```
func middlewareOne(next echo.HandlerFunc) echo.HandlerFunc {
    return func(c echo.Context) error {
        fmt.Println("from middleware one")
        return next(c)
    }
}

func middlewareTwo(next echo.HandlerFunc) echo.HandlerFunc {
    return func(c echo.Context) error {
        fmt.Println("from middleware two")
        return next(c)
    }
}
```

Registrasikan kedua middleware di atas. Kode di bawah ini adalah contoh cara registrasinya.

```
func main() {  
    e := echo.New()  
  
    e.Use(middlewareOne)  
    e.Use(middlewareTwo)  
  
    // ...
```

Jalankan aplikasi, lihat hasilnya.

```
[novalagung:chapter-b.6 $ go run 1-middleware.go
] / _/_/_\ / \ _\ / \
/ _/ / _/ - \ \ _\ \
/_/ \_/_/ _/_/\_ / v3.3.dev
High performance, minimalist Go web framework
https://echo.labstack.com
_____
0/ _____
0\_____
→ http server started on [::]:9000
from middleware one
from middleware two
threeeeee!
```

## C.6.2. Integrasi Middleware ber-skema Non-Echo-Middleware

Di echo, fungsi middleware harus memiliki skema `func(echo.HandlerFunc)echo.HandlerFunc`. Untuk 3rd party middleware, tetap bisa dikombinasikan dengan echo, namun membutuhkan sedikit penyesuaian tentunya.

Echo menyediakan solusi mudah untuk membantu integrasi 3rd party middleware, yaitu dengan menggunakan fungsi `echo.WrapMiddleware()` untuk mengkonversi middleware menjadi echo-compatible-middleware, dengan syarat skema harus dalam bentuk `func(http.Handler)http.Handler`.

Silakan praktikan kode berikut.

```
func middlewareSomething(next http.Handler) http.Handler {
    return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
        fmt.Println("from middleware something")
        next.ServeHTTP(w, r)
    })
}

func main() {
    e := echo.New()

    e.Use(echo.WrapMiddleware(middlewareSomething))

    // ...
}
```

Bisa dilihat, fungsi `middleware something` tidak menggunakan skema middleware milik echo, namun tetap bisa digunakan dalam `.use()` dengan cara dibungkus fungsi `echo.wrapMiddleware()`.

### C.6.3. Echo Middleware: Logger

Seperti yang sudah penulis jelaskan pada awal bab B, bahwa echo merupakan framework besar, didalamnya terdapat banyak dependency dan library, salah satunya adalah logging middleware.

Cara menggunakan logging middleware (ataupun middleware lainnya milik echo) adalah dengan meng-import package `github.com/labstack/echo/middleware`, lalu panggil nama middleware nya. Lebih detailnya silakan baca dokumentasi echo mengenai middleware di <https://echo.labstack.com/middleware>.

Berikut merupakan contoh penggunaan echo logging middleware.

```
package main

import (
    "github.com/labstack/echo"
    "github.com/labstack/echo/middleware"
    "net/http"
)

func main() {
    e := echo.New()

    e.Use(middleware.LoggerWithConfig(middleware.LoggerConfig{
        Format: "method=${method}, uri=${uri}, status=${status}\n",
    }))

    e.GET("/index", func(c echo.Context) (err error) {
        return c.JSON(http.StatusOK, true)
    })

    e.Logger.Fatal(e.Start(":9000"))
}
```

Cara menggunakan echo logging middleware adalah dengan membuat objek logging baru lewat statement `middleware.Logger()`, lalu membungkusnya dengan `e.Use()`. Atau bisa juga menggunakan `middleware.LoggerFactory()` jika logger yang dibuat memerlukan beberapa konfigurasi (tulis konfigurasinya sebagai property objek cetakan `middleware.LoggerFactory`, lalu tempatkan sebagai parameter method pemanggilan `.LoggerWithConfig()`).

Jalankan aplikasi, lalu lihat hasilnya.

Berikut merupakan list middleware yang disediakan oleh echo, atau cek <https://echo.labstack.com/middleware> untuk lebih detailnya.

- Basic Auth
  - Body Dump
  - Body Limit
  - CORS
  - CSRF
  - Casbin Auth
  - Gzip

- JWT
- Key Auth
- Logger
- Method Override
- Proxy
- Recover
- Redirect
- Request ID
- Rewrite
- Secure
- Session
- Static
- Trailing Slash

## C.6.4. 3rd Party Logging Middleware: Logrus

Selain dengan membuat middleware sendiri, ataupun menggunakan echo middleware, kita juga bisa menggunakan 3rd party middleware lain. Tinggal sesuaikan sedikit agar sesuai dengan skema fungsi middleware milik echo untuk bisa digunakan.

Next, kita akan coba untuk meng-implementasi salah satu golang library terkenal untuk keperluan logging, yaitu [logrus](#).

Buat file baru, import packages yang diperlukan, lalu buat fungsi `makeLogEntry()`, fungsi ini menerima satu parameter bertipe `echo.Context` dan mengembalikan objek logrus `*log.Entry`.

```
package main

import (
    "fmt"
    "github.com/labstack/echo"
    log "github.com/sirupsen/logrus"
    "net/http"
    "time"
)

func makeLogEntry(c echo.Context) *log.Entry {
    if c == nil {
        return log.WithFields(log.Fields{
            "at": time.Now().Format("2006-01-02 15:04:05"),
        })
    }

    return log.WithFields(log.Fields{
        "at": time.Now().Format("2006-01-02 15:04:05"),
        "method": c.Request().Method,
        "uri": c.Request().URL.String(),
        "ip": c.Request().RemoteAddr,
    })
}
```

Fungsi `makeLogEntry()` bertugas membuat basis log objek yang akan ditampilkan. Informasi standar seperti waktu, dibentuk di dalam fungsi ini. Khusus untuk log yang berhubungan dengan http request, maka informasi yang lebih detail dimunculkan (http method, url, dan IP).

Selanjutnya, buat fungsi `middlewareLogging()` dan `errorHandler()`.

```
func middlewareLogging(next echo.HandlerFunc) echo.HandlerFunc {
```

```

        return func(c echo.Context) error {
            makeLogEntry(c).Info("incoming request")
            return next(c)
        }
    }

    func errorHandler(err error, c echo.Context) {
        report, ok := err.(*echo.HTTPError)
        if ok {
            report.Message = fmt.Sprintf("http error %d - %v", report.Code, report.Message)
        } else {
            report = echo.NewHTTPError(http.StatusInternalServerError, err.Error())
        }

        makeLogEntry(c).Error(report.Message)
        c.HTML(report.Code, report.Message.(string))
    }
}

```

Fungsi `middlewareLogging()` bertugas untuk menampilkan log setiap ada http request masuk. Dari objek `*log.Entry` -yang-dicetak-lewat-fungsi- `makeLogEntry()` -, panggil method `Info()` untuk menampilkan pesan log dengan level adalah INFO.

Sedang fungsi `errorHandler` akan digunakan untuk meng-override default http error handler milik echo. Dalam fungsi ini log dengan level ERROR dimunculkan lewat pemanggilan method `Error()` milik `*log.Entry`.

Buat fungsi `main()`, implementasikan semua fungsi tersebut, siapkan yang harus disiapkan.

```

func main() {
    e := echo.New()

    e.Use(middlewareLogging)
    e.HTTPErrorHandler = errorHandler

    e.GET("/index", func(c echo.Context) error {
        return c.JSON(http.StatusOK, true)
    })

    lock := make(chan error)
    go func(lock chan error) { lock <- e.Start(":9000") }(lock)

    time.Sleep(1 * time.Millisecond)
    makeLogEntry(nil).Warning("application started without ssl/tls enabled")

    err := <-lock
    if err != nil {
        makeLogEntry(nil).Panic("failed to start application")
    }
}

```

Fungsi `main()` di atas berisikan beberapa kode yang jarang kita gunakan, pada saat men-start web server.

Web server di start dalam sebuah goroutine. Karena method `.Start()` milik echo adalah blocking, kita manfaatkan nilai baliknya untuk di kirim ke channel `lock`.

Selanjutnya dengan delay waktu 1 milidetik, log dengan level WARNING dimunculkan. Ini hanya simulasi saja, karena memang aplikasi tidak di start menggunakan ssl/tls. Dengan memberi delay 1 milidetik, maka log WARNING bisa muncul setelah log default dari echo muncul.

Nah pada bagian penerimaan channel, jika nilai baliknya tidak `nil` maka pasti terjadi error pada saat start web server, dan pada saat itu juga munculkan log dengan level PANIC.

OK, sekarang jalankan lalu test aplikasi.

```
[novalagung:chapter-b.6 $ go run 3-3rdparty-middleware-logrus.go
   _/ _/_/ _/ _/
  / _/_/_/_/_/_ \_
 /_/_/_/_/_/_/_/_/ v3.3.dev
High performance, minimalist Go web framework
https://echo.labstack.com
  _____ 0/
  0\_
=> http server started on [::]:9000
WARN[0000] application started without ssl/tls enabled
INFO[0003] incoming request
INFO[0011] incoming request
INFO[0015] incoming request
ERRO[0015] http error 404 - Not Found
INFO[0022] incoming request
ERRO[0022] http error 404 - http error 404 - Not Found
INFO[0024] incoming request
at="2018-07-13 19:49:54"
at="2018-07-13 19:49:57" ip="::1:57720" method=GET uri="/index
at="2018-07-13 19:50:05" ip="::1:57720" method=GET uri="/index
at="2018-07-13 19:50:09" ip="::1:57720" method=GET uri="/isthispageexists
at="2018-07-13 19:50:09" ip="::1:57720" method=GET uri="/isthispageexists
at="2018-07-13 19:50:16" ip="::1:57720" method=GET uri="/howaboutthispage
at="2018-07-13 19:50:16" ip="::1:57720" method=GET uri="/howaboutthispage
at="2018-07-13 19:50:18" ip="::1:57720" method=GET uri="/index
```

Satu kata, *cantik*.

- [Echo](#), by Vishal Rana (Lab Stack), MIT license
- [Logrus](#), by Simon Eskildsen, MIT license

## C.7. CLI Flag Parser (Kingpin v2)

Tidak jarang, sebuah aplikasi dalam eksekusinya membutuhkan argumen untuk disisipkan, entah itu mandatory atau tidak. Contohnya seperti berikut.

```
$ ./main --port=3000
```

Pada bab ini kita akan belajar cara parsing argumen eksekusi aplikasi. Parsing sebenarnya bisa dilakukan dengan cukup memainkan property `os.Args`. Tapi pada bab ini kita akan menggunakan 3rd party library [github.com/alecthomas/kingpin](https://github.com/alecthomas/kingpin) untuk mempermudah pelaksanaannya.

### C.7.1. Parsing Argument

Kita akan buat aplikasi yang bisa menerima bentuk argument seperti berikut.

```
# $ ./main ArgAppName <ArgPort>
$ ./main "My Application" 4000
```

Argument `ArgAppName` mandatory, harus diisi, sedangkan argument `ArgPort` adalah opsional (ada nilai default-nya).

OK, mari kita praktikan. Buat folder projek baru dengan isi satu buah main file. Siapkan dua buah property untuk menampung `appName` dan `port`, dan satu buah fungsi `main()`.

```
package main

import (
    "fmt"
    "net/http"

    "github.com/labstack/echo"
    "gopkg.in/alecthomas/kingpin.v2"
)

var (
    argAppName = kingpin.Arg("name", "Application name").Required().String()
    argPort    = kingpin.Arg("port", "Web server port").Default("9000").Int()
)

func main() {
    kingpin.Parse()

    // more code here ...
}
```

Statement `kingpin.Arg()` digunakan untuk menyiapkan objek penampung argument. Tulis nama argument sebagai parameter pertama, dan deskripsi argument sebagai parameter kedua. Informasi tersebut nantinya akan muncul ketika flag `--help` digunakan.

Untuk aplikasi yang memerlukan banyak argument, deklarasi variabel penampungnya harus dituliskan berurutan. Seperti contoh di atas `argAppName` merupakan argument pertama, dan `argPort` adalah argument kedua.

Chain statement `kingpin.Arg()` dengan beberapa method yang tersedia sesuai dengan kebutuhan. Berikut adalah penjelasan dari 4 method yang digunakan di atas.

- Method `.Required()` membuat argument yang ditulis menjadi mandatory. Jika tidak disisipkan maka muncul

error.

- Method `.String()` menandakan bahwa argument ditampung dalam tipe `string`.
- Method `.Default()` digunakan untuk menge-set default value dari argument. Method ini adalah kebalikan dari `.Required()`. Jika default value di-set maka argument boleh untuk tidak diisi. Objek penampung akan berisi default value.
- Method `.Int()` menandakan bahwa argument ditampung dalam tipe `int`.

Perlu diketahui, dalam pendefinisian argument, penulisan statement-nya harus diakhiri dengan pemanggilan method `.String()`, `.Int()`, `.Bool()`, atau method tipe lainnya yang di-support oleh kingpin. Lebih jelasnya silakan cek [laman dokumentasi](#).

Mari kita selesaikan aplikasi, silakan tulis kode berikut dalam fungsi `main()`.

```
appName := *argAppName
port := fmt.Sprintf(":%d", *argPort)

fmt.Printf("Starting %s at %s", appName, port)

e := echo.New()
e.GET("/index", func(c echo.Context) (err error) {
    return c.JSON(http.StatusOK, true)
})
e.Logger.Fatal(e.Start(port))
```

Objek argument kingpin pasti bertipe pointer, maka *dereference* objek tersebut untuk mengambil nilai aslinya.

Jalankan aplikasi, cek hasilnya.

```
[novalagung:chapter-b.7 $ go build 1-arg-parser.go
[novalagung:chapter-b.7 $ ./1-arg-parser --help
usage: 1-arg-parser [<flags>] <name> [<port>]

Flags:
  --help Show context-sensitive help (also try --help-long and --help-man).

Args:
  <name>    Application name
  [<port>]  Web server port

[novalagung:chapter-b.7 $ ./1-arg-parser "My Application" 3000
Starting My Application at :3000
 _/ _/_/ _/ _/
 / _/ _/ _ \_ \
 /_/_/_//_/_/ v3.3.dev
 High performance, minimalist Go web framework
 https://echo.labstack.com
 0/
 0\
 = http server started on [::]:3000
```

Bisa dilihat dari gambar di atas ketika flag `--help` dipanggil list semua argument muncul.

## C.7.2. Penggunaan Kingpin Application Instance

Dari yang sudah kita praktekan di atas, fungsi-fungsi diakses langsung dari package `kingpin`.

```
kingpin.Arg("name", "Application name").Required().String()
kingpin.Arg("port", "Web server port").Default("9000").Int()

kingpin.Parse()
```

Kingpin menyediakan fasilitas untuk membuat *-what-so-called-* objek kingpin application. Lewat objek ini, semua fungsi yang biasa digunakan (seperti `.Arg()` atau `.Parse()`) bisa diakses sebagai method.

Kelebihan menggunakan kingpin application, kita bisa buat custom handler untuk antisipasi error. Pada aplikasi yg sudah dibuat di atas, jika argument yang *required* tidak disisipkan dalam eksekusi binary, maka aplikasi langsung exit dan error muncul. Error sejenis ini bisa kita override jika menggunakan kingpin application.

Berikut adalah contoh implementasinya.

```
app      = kingpin.New("App", "Simple app")
argAppName = app.Arg("name", "Application name").Required().String()
argPort   = app.Arg("port", "Web server port").Default("9000").Int()

command, err := app.Parse(os.Args[1:])
if err != nil {
    // handler error here ...
}
```

Statement `kingpin.Arg()` diganti dengan `app.Arg()`. Juga, `kingpin.Parse()` diganti dengan `app.Parse()`, dengan pemanggilannya harus disisipkan `os.Args[1:]`.

Manfaatkan objek `err` kembalian `app.Parse()` untuk membuat custom error handling. Atau bisa tetap gunakan default custom error handling milik kingpin, caranya dengan membungkus statement `app.Parse()` ke dalam `kingpin.MustParse()`.

```
kingpin.MustParse(app.Parse(os.Args[1:]))
```

### C.7.3. Parsing Flag

Flag adalah argument yang lebih terstruktur. Golang sebenarnya sudah menyediakan package `flag`, isinya API untuk parsing flag.

- Contoh argument:

```
$ ./executable "My application" 4000
```

- Contoh flag:

```
$ ./executable --name="My application" --port=4000
$ ./executable --name "My application" --port 4000
$ ./executable --name "My application" -p 4000
```

Kita tetap menggunakan kingpin pada bagian ini. Pembuatan flag di kingpin tidak sulit, cukup gunakan `.Flag()` (tidak menggunakan `.Arg()`). Contohnya seperti berikut.

```
app      = kingpin.New("App", "Simple app")
flagAppName = app.Flag("name", "Application name").Required().String()
flagPort   = app.Flag("port", "Web server port").Short('p').Default("9000").Int()

kingpin.MustParse(app.Parse(os.Args[1:]))
```

Method `.Short()` digunakan untuk mendefinisikan short flag. Pada kode di atas, flag port bisa ditulis dalam bentuk `-port=value` ataupun `-p=value`.

Penggunaan flag `--help` akan memunculkan keterangan mendetail tiap-tiap flag.

Flag bisa dikombinasikan dengan argument.

```
[novalagung:chapter-b.7 $ go build 3-flag-parser.go
[novalagung:chapter-b.7 $ ./3-flag-parser --help
usage: App --name=NAME [<flags>]

Simple app

Flags:
      --help      Show context-sensitive help (also try --help-long and --help-man).
      --name=NAME  Application name
      -p, --port=9000 Web server port

[novalagung:chapter-b.7 $ ./3-flag-parser --name "My Application" -p 4500
Starting My Application at :4500
   _/\_ / \_ / \
  / \_ / \_ \_ \_ \
 / \_ \_ / \_ / \_ \
  v3.3.dev
High performance, minimalist Go web framework
https://echo.labstack.com
  _/ \_ / \_ / \_ \
  0/           0\
  0\           0\
⇒ http server started on [::]:4500
```

## C.7.4. Parsing Command

Command adalah bentuk yang lebih advance dari argument. Banyak command bisa dibuat, pendefinisan flag ataupun argument bisa dilakukan lebih spesifik, untuk masing-masing command.

Di bagian ini kita akan buat sebuah aplikasi untuk simulasi manajemen user. Tiga buah command dipersiapkan dengan skema sebagai berikut.

- Command `add`
  - Flag `--override`
  - Argument `user`
- Command `update`
  - Argument `old user`
  - Argument `new user`
- Command `delete`
  - Flag `--force`
  - Argument `user`

Mari kita praktekan, buat satu folder projek baru. Buat satu file main, isi dengan kode berikut.

```
package main

import (
    "fmt"
    "os"

    "gopkg.in/alecthomas/kingpin.v2"
)

var app = kingpin.New("App", "Simple app")
```

Method `.Command()` digunakan untuk membuat command. Pembuatan argument dan flag masing-masing command bisa dilakukan seperti biasanya, dengan mengakses method `.Arg()` atau `.Flag()`, hanya saja pengaksesannya lewat objek command masing-masing. Contoh:

```
var commandSomething = app.Command("something", "do something")
```

```
var commandSomethingArgX = commandSomething.Flag("x", "arg x").String()
var commandSomethingFlagY = commandSomething.Flag("y", "flag y").String()
```

OK, sekarang buat 3 command sesuai skema yang sudah disepakati di atas.

- Command add, beserta flag dan argument-nya.

```
var commandAdd = app.Command("add", "add new user")
var commandAddFlagOverride = commandAdd.Flag("override", "override existing user").
    Short('o').Bool()
var commandAddArgUser = commandAdd.Arg("user", "username").Required().String()
```

- Command update, beserta argument-nya.

```
var commandUpdate = app.Command("update", "update user")
var commandUpdateArgOldUser = commandUpdate.Arg("old", "old username").Required().
    String()
var commandUpdateArgNewUser = commandUpdate.Arg("new", "new username").Required().
    String()
```

- Command delete, beserta flag dan argument-nya.

```
var commandDelete      = app.Command("delete", "delete user")
var commandDeleteFlagForce = commandDelete.Flag("force", "force deletion").
    Short('f').Bool()
var commandDeleteArgUser  = commandDelete.Arg("user", "username").Required().
    String()
```

Buat fungsi main, lalu didalamnya siapkan action untuk masing-masing command. Gunakan method `.Action()` dengan parameter adalah fungsi ber-skema `func(*kingpin.ParseContext)error` untuk menambahkan action.

Di akhir, tulis statement untuk parsing.

```
func main() {
    commandAdd.Action(func(ctx *kingpin.ParseContext) error {
        // more code here ...
    })

    commandUpdate.Action(func(ctx *kingpin.ParseContext) error {
        // more code here ...
    })

    commandDelete.Action(func(ctx *kingpin.ParseContext) error {
        // more code here ...
    })

    Kingpin.MustParse(app.Parse(os.Args[1:]))
}
```

Berikut adalah isi masing-masing action dari ketiga command di atas.

- Action command `add` :

```
commandAdd.Action(func(ctx *kingpin.ParseContext) error {
    user := *commandAddArgUser
    override := *commandAddFlagOverride
    fmt.Printf("adding user %s, override %t\n", user, override)

    return nil
})
```

- Action command `update` :

```
commandUpdate.Action(func(ctx *kingpin.ParseContext) error {
    oldUser := *commandUpdateArgOldUser
    newUser := *commandUpdateArgNewUser
    fmt.Printf("updating user from %s %s \n", oldUser, newUser)

    return nil
})
```

- Action command `delete` :

```
commandDelete.Action(func(ctx *kingpin.ParseContext) error {
    user := *commandDeleteArgUser
    force := *commandDeleteFlagForce
    fmt.Printf("deleting user %s, force %t \n", user, force)

    return nil
})
```

Jalankan aplikasi untuk mengetes hasilnya.

Tambahkan flag `--help` pada pemanggilan command untuk menampilkan help command terpilih.

```
[novalagung:chapter-b.7 $ ./4-command-parser add --help
usage: App add [<flags>] <user>

add new user

Flags:
      --help      Show context-sensitive help (also try --help-long and --help-man).
      -o, --override  override existing user

Args:
      <user>  username

[novalagung:chapter-b.7 $ ./4-command-parser add --override "Noval Agung"
adding user Noval Agung, override true
novalagung:chapter-b.7 $ ]
```

Atau gunakan `--help-long` dalam eksekusi binary, untuk menampilkan help yang mendetail (argument dan flag tiap command juga dimunculkan).

```
[novalagung:chapter-b.7 $ ./4-command-parser --help-long
usage: App [<flags>] <command> [<args> ...]

Simple app

Flags:
  --help Show context-sensitive help (also try --help-long and --help-man).

Commands:
  help [<command>...]
  Show help.

  add [<flags>] <user>
  add new user

  -o, --override if user exists, then override

  update <old> <new>
  update user

  delete [<flags>] <user>
  delete user

  -f, --force force deletion

novalagung:chapter-b.7 $ ]
```

## C.7.5. Command Action Tanpa Menggunakan .Action()

Nilai balik statement `kingpin.MustParse()`, `kingpin.Parse()`, dan nilai balik pertama `app.Parse()` adalah sama, yaitu informasi command yang ditulis pada saat pemanggilan binary.

Dari informasi command tersebut, bisa kita kembangkan untuk membuat handler masing-masing command. Dengan ini tak perlu menggunakan method `.Action()` untuk menulis handler command. Contoh prakteknya seperti berikut.

```
commandInString := kingpin.MustParse(app.Parse(os.Args[1:]))
switch commandInString {

case commandAdd.FullCommand(): // add user
    user := *commandAddArgUser
    override := *commandAddFlagOverride
    fmt.Printf("adding user %s, override %t \n", user, override)

case commandUpdate.FullCommand(): // update user
    oldUser := *commandUpdateArgOldUser
    newUser := *commandUpdateArgNewUser
    fmt.Printf("updating user from %s %s \n", oldUser, newUser)

case commandDelete.FullCommand(): // delete user
    user := *commandDeleteArgUser
    force := *commandDeleteFlagForce
    fmt.Printf("deleting user %s, force %t \n", user, force)

}
```

## C.7.6. Advanced Command Line Application

Pembahasan di atas fokus tentang bagaimana cara parsing argument, flag, dan command yang disisipkan sewaktu eksekusi aplikasi. Aplikasi yang dieksekusi sendiri bisa berupa command-line-based ataupun web-based.

Jika pembaca ingin membuat aplikasi command line, penggunaan kingpin cukup membantu dalam proses pengembangan, tapi akan lebih mudah lagi jika menggunakan 3rd party library [Cobra](#).

Cobra merupakan library yang didesain khusus untuk development aplikasi berbasis command line. Library ini dibuat oleh author yang juga membuat kingpin.

---

- [Kingpin](#), by Alec Thomas, MIT license
- [Cobra](#), by Alec Thomas, MIT license

## C.8. Advanced Configuration File (Viper)

Pada bab ini kita akan belajar cara mudah parsing file konfigurasi menggunakan [Viper](#). Inti dari bab ini sebenarnya adalah sama dengan yang sudah dibahas pada bab [B-22 - Configuration](#), hanya saja disini proses parsing di-handle oleh 3rd party dengan tidak menggunakan struct untuk pengaksesannya.

Kekurangan dari teknik menyimpan konfigurasi dalam object struct adalah, pada saat ada kebutuhan untuk menambah atau merubah isi konfigurasi file, maka mengharuskan developer juga mengubah skema struct penampung.

### C.8.1. JSON Configuration

Mari langsung kita praktikan. Buat projek baru seperti biasa, buat file konfigurasi `app.conf.json`, isi dengan data berikut.

```
{
    "appName": "SimpleApp",
    "server": {
        "port": 9000
    }
}
```

Property `appName` berisi nama aplikasi, sedangkan `server.port` representasi dari port web server.

Selanjutnya buat `main.go`, lakukan parsing pada file konfigurasi.

```
package main

import (
    "fmt"
    "github.com/labstack/echo"
    "github.com/spf13/viper"
    "net/http"
)

func main() {
    e := echo.New()

    viper.SetConfigType("json")
    viper.AddConfigPath(".")
    viper.SetConfigName("app.config")

    err := viper.ReadInConfig()
    if err != nil {
        e.Logger.Fatal(err)
    }

    // ...
}
```

Kode di atas adalah contoh penggunaan dasar viper, untuk parsing file konfigurasi bertipe `JSON`. Fungsi `viper.SetConfigType()` digunakan untuk set jenis file konfigurasi.

Berikut merupakan list format yang didukung oleh viper.

- json

- toml
- yaml
- yml
- properties
- props
- prop

Fungsi `.AddConfigPath()` digunakan untuk mendaftarkan path folder dimana file-file konfigurasi berada. Fungsi ini bisa dipanggil beberapa kali, jika memang ada banyak file konfigurasi tersimpan dalam path berbeda.

Statement `.SetConfigName()` dieksekusi dengan parameter berisi nama file konfigurasi secara eksplisit tanpa ekstensi. Misalkan nama file adalah `app.config.json`, maka parameter cukup ditulis `app.config`.

Fungsi `.ReadInConfig()` digunakan untuk memproses file-file konfigurasi sesuai dengan path dan nama yang sudah ditentukan.

OK, kembali ke bagian tulis-menulis kode. Tambahkan beberapa kode untuk print nama aplikasi, sebuah rute, dan start web server.

```
fmt.Println("Starting", viper.GetString("appName"))

e.GET("/index", func(c echo.Context) (err error) {
    return c.JSON(http.StatusOK, true)
})

e.Logger.Fatal(e.Start(": " + viper.GetString("server.port")))
```

Cara pengaksesan konfigurasi bisa dilihat pada kode di atas. Statement `viper.GetString("appName")` mengembalikan string `"SimpleApp"`, sesuai dengan isi pada file konfigurasi.

Selain `.GetString()`, masih banyak lagi fungsi lain yang bisa digunakan, sesuaikan dengan tipe data property yang akan diambil.

Fungsi	Return type
Get(string)	interface{}
GetBool(string)	bool
GetDuration(string)	time.Duration
GetFloat64(string)	float64
GetInt(string)	int
GetInt32(string)	int32
GetInt64(string)	int64
GetSizeInBytes(string)	uint
GetString(string)	string
GetStringMap(string)	map[string]interface{}
GetStringMapString(string)	map[string]string
GetStringMapStringSlice(string)	map[string][]string
GetStringSlice(string)	[]string
GetTime(string)	time.Time

Pengaksesan property nested seperti `server.port` juga mudah, tinggal tulis saja skema property yang ingin diambil nilainya dengan separator tanda titik ( `.` ).

Jalankan aplikasi untuk test hasilnya.

```
[novalagung:chapter-B.8 $ go run 1-json-conf.go
Starting SimpleApp

      _/ \
     / \_ / \_ \
    / \_ / \_ / \_ \
   / \_ / \_ / \_ / \_ \
   v3.3.dev
High performance, minimalist Go web framework
https://echo.labstack.com
_____
0/
0\

⇒ http server started on [::]:9000
```

## C.8.2. YAML Configuration

Cara menggunakan viper pada file konfigurasi bertipe `.yaml` kurang lebih sama seperti pada file `.json`. Cukup ubah config type nya dan semua akan beres dengan sendirinya.

Mari kita langsung praktekan saja. Buat file konfigurasi baru `app.config.yaml` dengan isi berikut.

```
appName: SimpleApp
server:
  port: 9000
```

Pada bagian kode golang, cukup ubah argumen pemanggilan fungsi set config type.

```
viper.SetConfigType("yaml")
```

Jalankan aplikasi, dan hasilnya sama seperti sebelumnya.

## C.8.3. Watcher Configuration

Viper memiliki banyak fitur, satu diantaranya adalah mengaktifkan watcher pada file konfigurasi. Dengan adanya watcher, maka kita bisa membuat callback yang akan dipanggil setiap kali ada perubahan konfigurasi.

```
viper.WatchConfig()
viper.OnConfigChange(func(e fsnotify.Event) {
    fmt.Println("Config file changed:", e.Name)
})
```

Penggunaan fasilitas watcher memerlukan tambahan 3rd party library `fsnotify`.

- [Echo](#), by Vishal Rana (Lab Stack), MIT license
- [fsnotify](#), by fsnotify team, BSD-3-Clause license
- [Viper](#), by Steve Francia, MIT license



## C.9. Secure Cookie (Gorilla Securecookie)

Pada bab [B-21 HTTP Cookie](#), kita telah mempelajari tentang cookie dan implementasinya di golang.

Cookie memiliki beberapa atribut, diantaranya adalah `secure`. Dengan mengaktifkan atribut ini, informasi cookie menjadi lebih aman karena di-enkripsi, namun kapabilitas ini hanya akan aktif pada kondisi aplikasi SSL/TLS enabled.

TL;DR; Jika atribut `secure` di-isi `true`, namun web server TIDAK menggunakan SSL/TLS, maka cookie disimpan seperti biasa tanpa di-enkripsi.

Lalu bagaimana cara untuk membuat cookie aman pada aplikasi yang meng-enable SSL/TLS maupun yang tidak? caranya adalah dengan menambahkan step enkripsi data sebelum disimpan dalam cookie (dan men-decrypt data tersebut saat membaca).

Gorilla toolkit menyediakan library bernama `securecookie`, berguna untuk mempermudah enkripsi informasi cookie, dengan penerapan yang mudah. Pada bab ini kita akan mempelajari penggunaannya.

### C.9.1. Create & Read Secure Cookie

Penggunaan `securecookie` cukup mudah, buat objek secure cookie lewat `securecookie.New()` lalu gunakan objek tersebut untuk operasi encode-decode data cookie. Pemanggilan fungsi `.New()` memerlukan 2 buah argument.

- Hash key, diperlukan untuk otentikasi data cookie menggunakan algoritma kriptografi HMAC.
- Block key, adalah opsional, diperlukan untuk enkripsi data cookie. Default algoritma enkripsi yang digunakan adalah AES.

OK, langsung saja kita praktikan. Buat folder projek seperti biasa lalu isi `main.go` dengan kode berikut.

```
package main

import (
    "github.com/gorilla/securecookie"
    "github.com/labstack/echo"
    "github.com/novalagung/gubrak"
    "net/http"
    "time"
)

type M map[string]interface{}

var sc = securecookie.New([]byte("very-secret"), []byte("a-lot-secret-yay"))
```

Variabel `sc` adalah objek secure cookie. Objek ini kita gunakan untuk encode data yang akan disimpan dalam cookie, dan juga untuk decode data.

Buat fungsi `setCookie()`, bertugas untuk mempermudah pembuatan dan penyimpanan cookie.

```
func setCookie(c echo.Context, name string, data M) error {
    encoded, err := sc.Encode(name, data)
    if err != nil {
        return err
    }

    cookie := &http.Cookie{
        Name:     name,
        Value:    encoded,
        Path:    "/",
```

```

        Secure: false,
        HttpOnly: true,
        Expires: time.Now().Add(1 * time.Hour),
    }
http.SetCookie(c.Response(), cookie)

return nil
}

```

Method `sc.Encode()` digunakan untuk encoding data dengan identifier adalah isi variabel `name`. Variabel `encoded` menampung data setelah di-encode, lalu variabel ini dimasukan ke dalam objek cookie.

Cara menyimpan cookie masih sama, menggunakan `http.SetCookie`.

Selanjutnya buat fungsi `getCookie()`, untuk mempermudah proses pembacaan cookie yang tersimpan.

```

func getCookie(c echo.Context, name string) (M, error) {
    cookie, err := c.Request().Cookie(name)
    if err == nil {

        data := M{}
        if err = sc.Decode(name, cookie.Value, &data); err == nil {
            return data, nil
        }
    }

    return nil, err
}

```

Setelah cookie diambil menggunakan `c.Request().Cookie()`, data didalamnya perlu di-decode agar bisa terbaca.

Method `sc.Decode()` digunakan untuk decoding data.

OK, sekarang buat fungsi `main()`, lalu isi dengan kode di bawah ini.

```

const COOKIE_NAME = "data"

e := echo.New()

e.GET("/index", func(c echo.Context) error {
    data, err := getCookie(c, COOKIE_NAME)
    if err != nil && err != http.ErrNoCookie && err != securecookie.ErrMacInvalid {
        return err
    }

    if data == nil {
        data = M{"Message": "Hello", "ID": gubrak.RandomString(32)}

        err = setCookie(c, COOKIE_NAME, data)
        if err != nil {
            return err
        }
    }

    return c.JSON(http.StatusOK, data)
})

e.Logger.Fatal(e.Start(":9000"))

```

Konstanta `COOKIE_NAME` disiapkan, kita gunakan sebagai identifier cookie. Dan sebuah rute juga disiapkan dengan tugas menampilkan data cookie jika sudah ada, dan membuat cookie baru jika belum ada.

Dalam handler rute, terdapat beberapa proses terjadi. Pertama, objek cookie dengan identifier `COOKIE_NAME` diambil, jika muncul error, dan jenisnya adalah selain error karena cookie tidak ada, dan error-nya selain *invalid cookie*, maka kembalikan objek error tersebut.

`http.ErrNoCookie` adalah variabel penanda error karena cookie kosong, sedangkan `securecookie.ErrMacInvalid` adalah representasi dari invalid cookie.

Lalu, kita cek data cookie yang dikembalikan, jika kosong (bisa karena cookie belum dibuat ataupun sudah ada tetapi datanya kosong) maka buat data baru untuk disimpan dalam cookie. Data tersebut bertipe `map`, salah satu elemen map tersebut ada yg value-nya adalah random.

Pada kode di atas, generate random string dilakukan dengan memanfaatkan 3rd party library [gubrak](#).

Pengaksesan rute akan memunculkan data yang sama. Karena pembuatan cookie hanya dilakukan ketika datanya kosong atau cookie nya belum dibuat.

Jalankan aplikasi untuk mengetes hasilnya. Lakukan refresh beberapa kali, data yang muncul pasti sama.



Lihat pada response header url `index`, data pada cookie terlihat sudah dalam kondisi encoded dan encrypted.

Name	x Headers	Preview	Response	Cookies	Timing
Index	<p>General</p> <p>Response Headers (3)</p> <p>Request Headers view source</p> <pre>Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8 Accept-Encoding: gzip, deflate, br Accept-Language: en-US,en;q=0.9 Cache-Control: max-age=0 Connection: keep-alive Cookie: data=MTUzMTcxNzE0NHxqYTByRENVeF9hUnpIaDBhTGJWnNJb0lrlrE9ZSVZ6Mvhzd200d3I1elhIUTZUDZmb094UTRuM19aT0pKTUtaNndhQkc3TFoza2xONGF1TkVfUUxZdTdrZU9pdJfSxdwZ05PT05Da2ZmajJLZTBYcF9LbDhj0GU0R1pj0ER1ld0lcQUpmsDA9fp016dLCI0dTzaxTgLwsr8wdzeAVH5tANZEfVqdvIU Host: localhost:9000 Upgrade-Insecure-Requests: 1 User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_5) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/67.0.3396.99 Safari/537.36</pre>				

## C.9.2. Delete Secure Cookie

Securecookie perannya hanya pada bagian encode-decode data cookie, sedangkan proses simpan baca cookie masih sama seperti penerapan cookie biasa. Maka cara menghapus cookie pun masih sama, yaitu dengan meng-expired-kan cookie yang sudah disimpan.

```
cookie := &http.Cookie{}
cookie.Name = name
cookie.Path = "/"
cookie.MaxAge = -1
cookie.Expires = time.Unix(0, 0)
http.SetCookie(c.Response(), cookie)
```

- [Echo](#), by Vishal Rana (Lab Stack), MIT license
- [Gorilla Securecookie](#), by Gorilla web toolkit team, BSD-3-Clause license
- [Gubrak](#), by Noval Agung, MIT license



## C.10. Session (Gorilla Session)

Session adalah sebuah konsep penyimpanan data yang shared antar http request. Session umumnya menggunakan cookie untuk menyimpan identifier (kita sebut sebagai **SessionID**). Informasi SessionID tersebut ber-asosiasi dengan data (kita sebut sebagai **SessionData**) yang disimpan di sisi back end dalam media tertentu.

Di back end, SessionData disimpan dalam media database, atau memory, atau fasilitas penyimpanan lainnya. Bisa saja sebenarnya jika SessionData juga disimpan dalam cookie, dengan memanfaatkan secure cookie maka SessionData tersebut ter-enkripsi dan aman dari peretas. Memang aman, tapi jelasnya lebih aman kalau disimpan di sisi server.

Pada bab ini kita akan mempelajari penerapan session di golang menggunakan beberapa jenis media penyimpanan, yaitu mongo db, postgres sql db, dan secure cookie.

### C.10.1. Manage Session Menggunakan Gorilla Sessions

[Gorilla Sessions](#) adalah library untuk manajemen session di golang.

Gorilla menyediakan interface `sessions.Store`, lewat interface ini kita bisa mengakses 3 buah method penting untuk manage session. Store sendiri adalah representasi dari media penyimpanan di back end, bisa berupa database, memory, atau lainnya. Objek store dibuat oleh library lain yang merupakan implementasi dari interface store itu sendiri.

Kembali ke pembahasan mengenai store, 3 buah method yang dimaksud adalah berikut:

- Method `.Get(r *http.Request, name string) (*Session, error)`, mengembalikan objek session. Jika session yang dengan `name` yang dicari tidak ada, maka objek session baru dikembalikan.
- Method `.New(r *http.Request, name string) (*Session, error)`, mengembalikan objek session baru.
- Method `.Save(r *http.Request, w http.ResponseWriter, s *Session) error`, digunakan untuk menyimpan session baru.

Dari ketiga method di atas saya rasa cukup jelas sekilas bagaimana cara mengakses, membuat, dan menyimpan session.

Kita akan fokus membahas API milik interface `sessions.Store` dahulu, mengenai pembuatan store sendiri ada di pembahasan setelahnya.

Lalu bagaimana dengan operasi hapus/delete? Seperti yang sudah dijelaskan sebelumnya, informasi session dipisah menjadi dua, pertama adalah SessionID yang disimpan di cookie, dan kedua adalah SessionData yang disimpan di back end. Cara untuk menghapus session adalah cukup dengan meng-expired-kan cookie yang menyimpan SessionID.

Cookie merupakan salah satu header pada http request, operasi yang berhubungan dengan cookie pasti membutuhkan objek `http.Request` dan `http.ResponseWriter`. Jika menggunakan echo, kedua objek tersebut bisa diakses lewat objek `http context echo.Context`.

### C.10.2. Membuat Objek Session Baru

Berikut adalah contoh cara membuat session lewat store.

```
e.GET("/set", func(c echo.Context) error {
    session, _ := store.Get(c.Request(), SESSION_ID)
```

```

    session.Values["message1"] = "hello"
    session.Values["message2"] = "world"
    session.Save(c.Request(), c.Response())

    return c.Redirect(http.StatusTemporaryRedirect, "/get")
)

```

Statement `store.Get()` mengembalikan dua objek dengan tipe `session.Session` dan `error`. Pemanggilan method ini memerlukan dua buah parameter untuk disisipkan, yaitu objek http request, dan nama/key SessionID yang disiapkan di konstanta `SESSION_ID`. Method `.Get()` ini akan selalu mengembalikan objek session, ada ataupun tidak ada session yang dicari, objek session tetap dikembalikan.

Pembuatan objek session baru bisa dilakukan lewat `store.New()` maupun `store.Get()`.

Dari objek session, akses property mutable `.values` untuk mengambil ataupun mengisi data session. Objek ini bertipe `map[interface{}]interface{}`, berarti SessionData yang akan disimpan juga harus memiliki identifier.

Pada contoh di atas, dua buah data bertipe string disimpan, dengan identifier data yang juga string.

- SessionData `"hello"` disimpan dengan identifier adalah `message1`.
- SessionData `"world"` disimpan dengan identifier adalah `message2`.

Cara menyimpan session adalah dengan memanggil method `.Save()` milik objek session, dengan parameter adalah http request dan response.

### C.10.3. Mengakses SessionData

SessionData diakses dari objek session, berikut merupakan contoh caranya.

```

e.GET("/get", func(c echo.Context) error {
    session, _ := store.Get(c.Request(), SESSION_ID)

    if len(session.Values) == 0 {
        return c.String(http.StatusOK, "empty result")
    }

    return c.String(http.StatusOK, fmt.Sprintf(
        "%s %s",
        session.Values["message1"],
        session.Values["message2"],
    ))
}

```

Seperti yang sudah dibahas di atas, objek `session` kembalian `store.Get()` TIDAK akan pernah berisi `nil`. Ada atau tidak, objek session selalu dikembalikan.

Dari objek session dilakukan pengecekan ada tidaknya SessionData, caranya dengan cara menghitung isi property `.values` yang tipenya `map`. Jika isinya kosong maka session belum ada (atau mungkin ada hanya saja expired, atau bisa saja ada tapi invalid).

Pada kode di atas, jika SessionData kosong maka string `empty result` ditampilkan ke layar. Sedangkan jika ada, maka kedua SessionData (`message1` dan `message2`) diambil lalu ditampilkan.

### C.10.4. Menghapus Session

Cara menghapus session adalah dengan meng-expired-kan max age cookie-nya. Property max age bisa diakses lewat `session.Options.MaxAge`.

```

e.GET("/delete", func(c echo.Context) error {
    session, _ := store.Get(c.Request(), SESSION_ID)
    session.Options.MaxAge = -1
    session.Save(c.Request(), c.Response())

    return c.Redirect(http.StatusTemporaryRedirect, "/get")
})

```

Isi dengan `-1` agar expired, lalu simpan ulang kembali session-nya.

## C.10.5. Session Store dan Context Clear Handler

Session Store adalah representasi dari media tempat dimana data asli session disimpan. Gorilla menyediakan `CookieStore`, penyimpanan data asli pada store ini adalah juga di dalam cookie, namun di-encode dan di-enkripsi menggunakan `Securecookie`.

Selain `CookieStore`, ada banyak store lain yang bisa kita gunakan. Komunitas begitu baik telah menyediakan berbagai macam store berikut.

- [github.com/starJammer/gorilla-sessions-arangodb](https://github.com/starJammer/gorilla-sessions-arangodb) - ArangoDB
- [github.com/yosssi/boltstore](https://github.com/yosssi/boltstore) - Bolt
- [github.com/srinathgs/couchbasestore](https://github.com/srinathgs/couchbasestore) - Couchbase
- [github.com/denizeren/dynamostore](https://github.com/denizeren/dynamostore) - Dynamodb on AWS
- [github.com/savaki/dynastore](https://github.com/savaki/dynastore) - DynamoDB on AWS (Official AWS library)
- [github.com/bradleypeabody/gorilla-sessions-memcache](https://github.com/bradleypeabody/gorilla-sessions-memcache) - Memcache
- [github.com/dsoprea/go-appengine-sessioncascade](https://github.com/dsoprea/go-appengine-sessioncascade) - Memcache/Datastore/Context in AppEngine
- [github.com/kidstuff/mongostore](https://github.com/kidstuff/mongostore) - MongoDB
- [github.com/srinathgs/mysqlstore](https://github.com/srinathgs/mysqlstore) - MySQL
- [github.com/EnumApps/clustersqlstore](https://github.com/EnumApps/clustersqlstore) - MySQL Cluster
- [github.com/antonlindstrom/pgstore](https://github.com/antonlindstrom/pgstore) - PostgreSQL
- [github.com/boj/redistore](https://github.com/boj/redistore) - Redis
- [github.com/boj/rethinkstore](https://github.com/boj/rethinkstore) - RethinkDB
- [github.com/boj/riakstore](https://github.com/boj/riakstore) - Riak
- [github.com/michaeljs1990/sqlitestore](https://github.com/michaeljs1990/sqlitestore) - SQLite
- [github.com/wader/gormstore](https://github.com/wader/gormstore) - GORM (MySQL, PostgreSQL, SQLite)
- [github.com/gernest/qlstore](https://github.com/gernest/qlstore) - ql
- [github.com/quasoft/memstore](https://github.com/quasoft/memstore) - In-memory implementation for use in unit tests
- [github.com/lafriks/xormstore](https://github.com/lafriks/xormstore) - XORM (MySQL, PostgreSQL, SQLite, Microsoft SQL Server, TiDB)

Objek store dibuat sekali di awal (atau bisa saja berkali-kali di tiap handler, tergantung kebutuhan). Pada pembuatan objek store, umumnya ada beberapa konfigurasi yang perlu disiapkan dan dua buah keys: authentication key dan encryption key.

Dari objek store tersebut, dalam handler, kita bisa mengakses objek session dengan menyisipkan context `http request`. Silakan lihat kode berikut untuk lebih jelasnya. Store direpresentasikan oleh variabel objek `store`.

```

package main

import (
    "fmt"
    "github.com/gorilla/context"
    "github.com/gorilla/sessions"
    "github.com/labstack/echo"
    "net/http"
)

```

```

const SESSION_ID = "id"

func main() {
    store := newMongoStore()

    e := echo.New()

    e.Use(echo.WrapMiddleware(context.ClearHandler))

    e.GET("/set", func(c echo.Context) error {
        session, _ := store.Get(c.Request(), SESSION_ID)
        session.Values["message1"] = "hello"
        session.Values["message2"] = "world"
        session.Save(c.Request(), c.Response())

        return c.Redirect(http.StatusTemporaryRedirect, "/get")
    })

    // ...
}

```

Sesuai dengan README Gorilla Session, library ini jika digabung dengan library lain selain gorilla mux, akan berpotensi menyebabkan memory leak. Untuk mengcover isu ini maka middleware `context.ClearHandler` perlu diregistrasikan. Middleware tersebut berada dalam library [Gorilla Context](#).

## C.10.6. Mongo DB Store

Kita akan mempelajari pembuatan session store dengan media adalah mongo db. Sebelum kita mulai, ada dua library yang perlu di `go get` .

- [gopkg.in/mgo.v2](https://github.com/golang/mgo.v2)
- [github.com/kidstuff/mongostore](https://github.com/kidstuff/mongostore)

Library pertama, `mgo.v2` merupakan driver mongo db untuk golang. Koneksi dari golang ke mongodb akan kita buat lewat API library ini.

Library kedua, merupakan implementasi dari interface `sessions.Store` untuk mongo db.

Silakan kombinasikan semua coding yang sudah kita tulis di atas agar menjadi satu aplikasi. Lalu buat fungsi `newMongoStore()` .

```

import (
    "fmt"
    "github.com/gorilla/context"
    "github.com/kidstuff/mongostore"
    "github.com/labstack/echo"
    "gopkg.in/mgo.v2"
    "log"
    "net/http"
    "os"
)

// ...

func newMongoStore() *mongostore.MongoStore {
    mgoSession, err := mgo.Dial("localhost:27123")
    if err != nil {
        log.Println("ERROR", err)
        os.Exit(0)
    }

    dbCollection := mgoSession.DB("learnwebgolang").C("session")
    maxAge := 86400 * 7
    ensureTTL := true
}

```

```

authKey := []byte("my-auth-key-very-secret")
encryptionKey := []byte("my-encryption-key-very-secret123")

store := mongostore.NewMongoStore(
    dbCollection,
    maxAge,
    ensureTTL,
    authKey,
    encryptionKey,
)
return store
}

```

Statement `mgo.Dial()` digunakan untuk terhubung dengan mongo db server. Method dial mengembalikan dua objek, salah satunya adalah mgo session.

Pada saat pembuatan buku ini, penulis menggunakan mongo db server yang up pada port `27123`, silakan menyesuaikan connection string dengan credentials mongo db yang digunakan.

Dari mgo session akses database lewat method `.DB()`, lalu akses collection yang ingin digunakan sebagai media penyimpanan data asli session lewat method `.C()`.

Statement `mongostore.NewMongoStore()` digunakan untuk membuat mongo db store. Ada beberapa parameter yang diperlukan: objek collection mongo di atas, dan dua lagi lainnya adalah authentication key dan encryption key.

Jika pembaca merasa bingung, silakan langsung buka source code untuk [bab B.10 ini di github](#), mungkin membantu.

## C.10.7. Postgres SQL Store

Pembuatan postgres store caranya kurang lebih sama dengan mongo store. Library yang dipakai adalah [github.com/antonlindstrom/pgstore](https://github.com/antonlindstrom/pgstore).

Gunakan `pgstore.NewPGStore()` untuk membuat store. Isi parameter pertama dengan connection string postgres server, lalu authentication key dan encryption key.

```

import (
    "fmt"
    "github.com/antonlindstrom/pgstore"
    "github.com/gorilla/context"
    "github.com/labstack/echo"
    "log"
    "net/http"
    "os"
)
// ...

func newPostgresStore() *pgstore.PGStore {
    url := "postgres://novalagung:@127.0.0.1:5432/novalagung?sslmode=disable"
    authKey := []byte("my-auth-key-very-secret")
    encryptionKey := []byte("my-encryption-key-very-secret123")

    store, err := pgstore.NewPGStore(url, authKey, encryptionKey)
    if err != nil {
        log.Println("ERROR", err)
        os.Exit(0)
    }
    return store
}

```

## C.10.8. Secure Cookie Store

Penggunaan cookie store kurang penulis anjurkan, meski sebenarnya cukup aman. Implementasi store jenis ini adalah yang paling mudah, karena tidak butuh database server atau media lainnya; dan juga karena API untuk cookie store sudah tersedia dalam gorilla sessions secara default.

```
import (
    "fmt"
    "github.com/gorilla/context"
    "github.com/gorilla/sessions"
    "github.com/labstack/echo"
    "net/http"
)

// ...

func newCookieStore() *sessions.CookieStore {
    authKey := []byte("my-auth-key-very-secret")
    encryptionKey := []byte("my-encryption-key-very-secret123")

    store := sessions.NewCookieStore(authKey, encryptionKey)
    store.Options.Path = "/"
    store.Options.MaxAge = 86400 * 7
    store.Options.HttpOnly = true

    return store
}
```

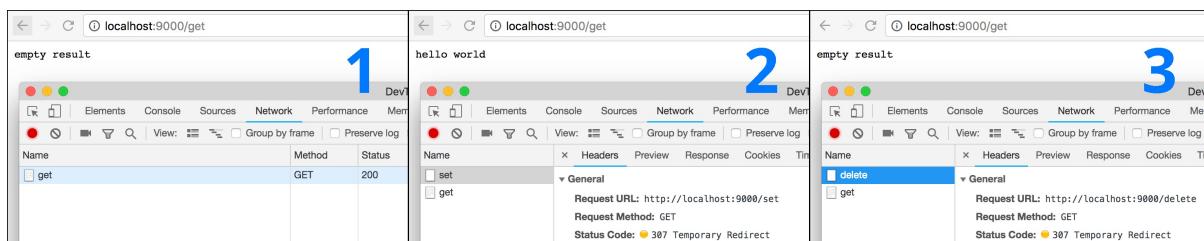
Tentukan path dan default max age cookie lewat `store.Options`.

## C.10.9. Test Aplikasi

Silakan gabung semua kode yang sudah kita pelajari (kecuali bagian store), lalu pilih salah satu implementasi store di atas. Jalankan aplikasi untuk testing.

Tujuan dari kode yang kita tulis kurang lebih sebagai berikut.

1. Ketika `/get` diakses untuk pertama kali, `empty result` muncul, tidak ada data session yang disimpan sebelumnya.
2. Rute `/set` diakses, lalu sebuah session disimpan, dari rute ini pengguna di-redirect ke `/get`, sebuah pesan muncul yang sumber datanya tak lain adalah dari session.
3. Rute `/delete` diakses, session dihapus, lalu di-redirect lagi ke `/get`, pesan `empty result` muncul kembali karena session sudah tidak ada (dihapus).



- [Echo](#), by Vishal Rana (Lab Stack), MIT license
- [Gorilla Sessions](#), by Gorilla web toolkit team, BSD-3-Clause license
- [Gorilla Context](#), by Gorilla web toolkit team, BSD-3-Clause license
- [Gorilla Securecookie](#), by Gorilla web toolkit team, BSD-3-Clause license

- [PG Store](#), by Anton Lindström, MIT License
- [Mongo Store](#), by Nguyễn Văn Cao Nguyên, BSD-3-Clause License
- [Mgo v2, Golang Mongo Driver](#), by Gustavo Niemeyer, Simplified BSD License

## C.12. CORS & Preflight Request

Pada bab ini kita akan belajar tentang Cross-Origin Resource Sharing (CORS) dan Preflight Request.

### C.12.1. Teori & Penerapan

CORS adalah mekanisme untuk memberi tahu browser, apakah sebuah request yang di-dispatch dari aplikasi web domain lain atau origin lain, ke aplikasi web kita itu diperbolehkan atau tidak. Jika aplikasi kita tidak mengijinkan maka akan muncul error, dan request pasti digagalkan oleh browser.

CORS hanya berlaku pada request-request yang dilakukan lewat browser, dari javascript; dan tujuan request-nya berbeda domain/origin. Jadi request yang dilakukan dari curl maupun dari back end, tidak terkena dampak aturan CORS.

Request jenis ini biasa disebut dengan istilah cross-origin HTTP request.

Konfigurasi CORS dilakukan di **response header** aplikasi web. Penerapannya di semua bahasa pemrograman yang web-based adalah sama, yaitu dengan memanipulasi response header-nya. Berikut merupakan list header yang bisa digunakan untuk konfigurasi CORS.

- Access-Control-Allow-Origin
- Access-Control-Allow-Methods
- Access-Control-Allow-Headers
- Access-Control-Allow-Credentials
- Access-Control-Max-Age

Konfigurasi CORS berada di sisi server, di aplikasi web tujuan request.

Permisalan: aplikasi kita di local mengambil data dari google.com, maka konfigurasi CORS berada di google.com; Jika kita terkena error CORS maka tak ada lagi yang bisa dilakukan, karena CORS aplikasi tujuan dikontrol oleh orang-orang yang ada di google.com.

Agar lebih mudah untuk dipahami bagaimana penerapannya, mari langsung kita praktikan seperti biasanya.

### C.12.2. Aplikasi dengan konfigurasi CORS sederhana

Buat projek baru, lalu isi fungsi `main()` dengan kode berikut. Aplikasi sederhana ini akan kita jalankan pada domain atau origin `http://localhost:3000/`, lalu akan kita coba akses dari domain berbeda.

```
package main

import (
    "log"
    "net/http"
)

func main() {
    http.HandleFunc("/index", func(w http.ResponseWriter, r *http.Request) {
        w.Header().Set("Access-Control-Allow-Origin", "https://www.google.com")
        w.Header().Set("Access-Control-Allow-Methods", "OPTIONS, GET, POST, PUT")
        w.Header().Set("Access-Control-Allow-Headers", "Content-Type, X-CSRF-Token")

        if r.Method == "OPTIONS" {
            w.Write([]byte("allowed"))
            return
        }
    })
}
```

```

        }

        w.Write([]byte("hello"))
    })

    log.Println("Starting app at :9000")
    http.ListenAndServe(":9000", nil)
}

```

Seperti yang sudah dijelaskan, bahwa konfigurasi CORS berada di header response. Pada kode di atas 3 buah property header untuk keperluan CORS digunakan.

Header `Access-Control-Allow-Origin` digunakan untuk menentukan domain mana saja yang diperbolehkan mengakses aplikasi ini. Kita bisa set value-nya dengan banyak origin, hal ini diperbolehkan dalam [spesifikasi CORS](#) namun sayangnya banyak browser yang tidak support.

```
Access-Control-Allow-Origin: https://www.google.com
```

Kode di atas artinya request yang di-dispatch dari <https://www.google.com> diijinkan untuk masuk; Penulis memilih domain google karena testing akan dilakukan dari sana, dengan tujuan destinasi request adalah

```
http://localhost:3000/ .
```

Simulasi pada bab ini adalah **aplikasi web localhost:3000 diakses dari google.com** (eksekusi request sendiri kita lakukan dari browser dengan memanfaatkan developer tools milik chrome). BUKAN google.com diakses dari aplikasi web localhost:3000, jangan sampai dipahami terbalik.

Kembali ke pembahasan source code. Dua header CORS lainnya digunakan untuk konfigurasi yang lebih mendetail.

```
Access-Control-Allow-Methods: OPTIONS, GET, POST, PUT
Access-Control-Allow-Headers: Content-Type, X-CSRF-Token
```

Header `Access-Control-Allow-Methods` menentukan HTTP Method mana saja yang diperbolehkan masuk (penulisannya dengan pembatas koma).

Dianjurkan untuk selalu memasukan method `OPTIONS` karena method ini dibutuhkan oleh preflight request.

Header `Access-Control-Allow-Headers` menentukan key header mana saja yang diperbolehkan di-dalam request.

Jika request tidak memenuhi salah satu saja dari ke-tiga rules di atas, maka request bisa dipastikan gagal. Contoh:

- Request dari <https://novalagung.com> ke <http://localhost:3000>, hasilnya pasti gagal, karena origin novalagung.com tidak diijinkan untuk mengakses <http://localhost:3000>.
- Request dari <https://www.google.com> ke <http://localhost:3000> dengan method adalah `DELETE`, hasilnya pasti gagal. Method `DELETE` adalah tidak di-ijinkan. hanya empat method `OPTIONS` , `GET` , `POST` , `PUT` yang diijinkan.
- Request dari <https://www.google.com> ke <http://localhost:3000> dengan method `GET`, dengan header `Authorization: Basic xxx` dan `X-CSRF-Token: xxxx` , hasilnya adalah gagal. Karena salah satu dari kedua header tersebut tidak diijinkan (header `Authorization` ).
- Request dari <https://www.google.com> ke <http://localhost:3000> dengan method `GET` dan memiliki header `Content-Type` adalah diijinkan masuk, karena memenuhi semua aturan yang kita definiskan.

Khusus untuk beberapa header seperti `Accept` , `Origin` , `Referer` , dan `User-Agent` tidak terkena efek CORS, karena header-header tersebut secara otomatis di-set di setiap request.

### C.12.3. Testing CORS

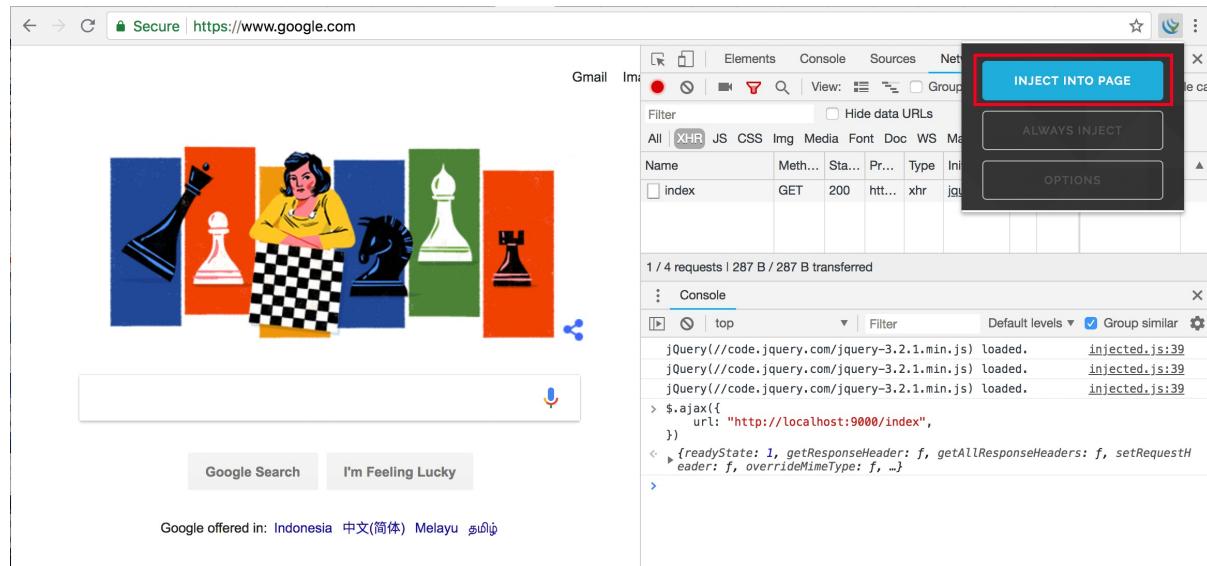
- **Persiapan**

Ada beberapa hal yang perlu dipersiapkan. Pertama, pastikan punya google chrome. Lalu install extension [jQuery Injector](#). Buka <https://www.google.com> lalu inject jQuery. Dengan melakukan inject jQuery secara paksa maka dari situs google kita bisa menggunakan jQuery.

Buka chrome developer tools, klik tab console. Lalu jalankan perintah jQuery AJAX berikut.

```
$.ajax({
  url: "http://localhost:9000/index",
})
```

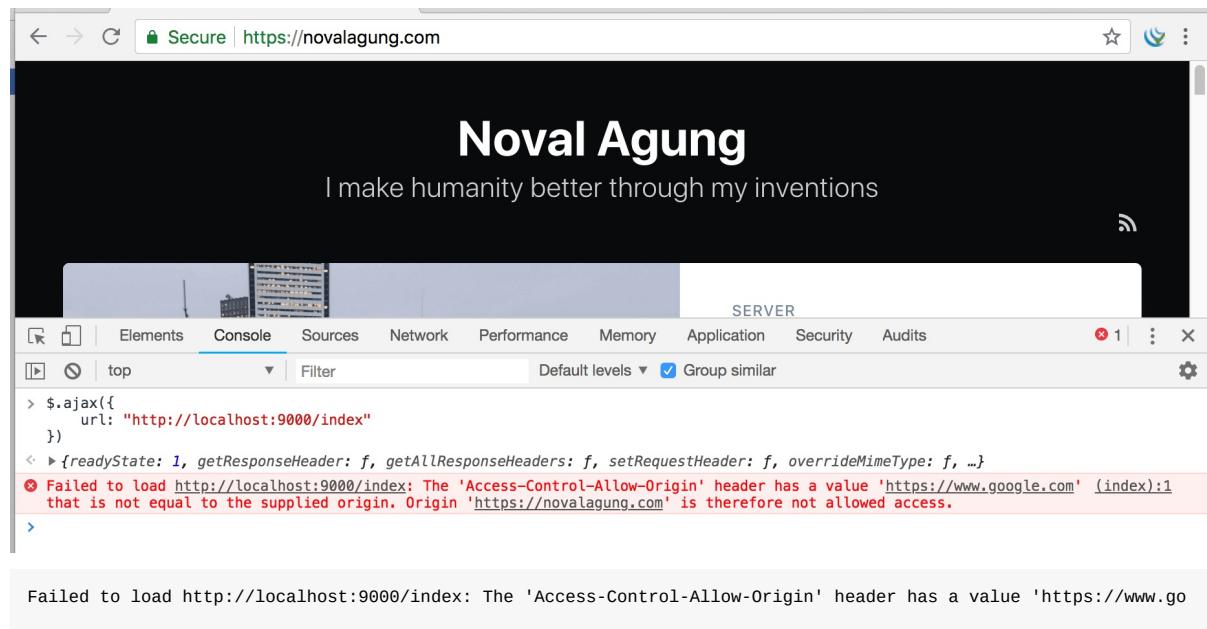
Silakan lihat gambar berikut untuk memperjelas.



Bisa dilihat, tidak ada error, karena memang request dari google diijinkan. Silakan coba-coba melakukan request AJAX lainnya dengan method POST, DELETE, atau lainnya; atau ditambah dengan menyisipkan header tertentu dalam ajax request.

### • Akses <http://localhost:9000> dari Origin yang Tidak Didasarkan di CORS

Selanjutnya coba buka tab baru, buka <https://novalagung.com>, lalu jalankan script yang sama.



```
ogle.com' that is not equal to the supplied origin. Origin 'https://novalagung.com' is therefore not allowed access.
```

Dari screenshot dan error log di atas, bisa dilihat bahwa request gagal. Hal ini dikarenakan origin <https://novalagung.com> tidak diijinkan untuk mengakses <http://localhost:9000>.

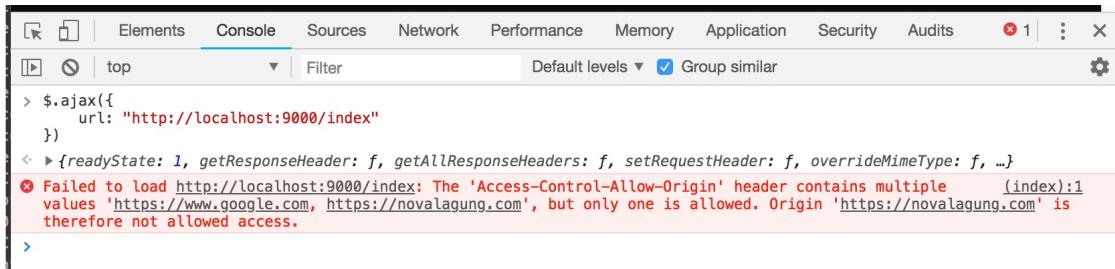
### • CORS Multiple Origin

Sekarang coba tambahkan situs <https://novalagung.com> ke CORS header.

```
http.HandleFunc("/index", func(w http.ResponseWriter, r *http.Request) {
    w.Header().Set("Access-Control-Allow-Origin",
        "https://www.google.com, https://novalagung.com")

    // ...
})
```

Jalankan ulang aplikasi, lalu dispatch lagi AJAX dari situs tersebut.



Masih tetap error, tapi berbeda dengan error sebelumnya.

```
Failed to load http://localhost:9000/index: The 'Access-Control-Allow-Origin' header contains multiple values 'https://www.google.com, https://novalagung.com', but only one is allowed. Origin 'https://novalagung.com' is therefore not allowed access.
```

Sebenarnya sudah kita singgung juga di atas, bahwa di spesifikasi adalah diperbolehkan isi header `Access-Control-Allow-Origin` lebih dari satu website. Namun, kebanyakan browser tidak mendukung bagian ini. Oleh karena itu error di atas muncul. Konfigurasi ini termasuk tidak valid, hasilnya kedua website tersebut tidak punya ijin masuk.

### • Allow All

Gunakan tanda asteriks (`*`) sebagai nilai ketiga CORS header untuk memberi ijin ke semua.

```
// semua origin mendapat ijin akses
w.Header().Set("Access-Control-Allow-Origin", "*")

// semua method diperbolehkan masuk
w.Header().Set("Access-Control-Allow-Methods", "*")

// semua header diperbolehkan untuk disisipkan
w.Header().Set("Access-Control-Allow-Headers", "*")
```

## C.12.4. Preflight Request

### • Teori

Dalam konteks CORS, request dikategorikan menjadi 2 yaitu, **Simple Request** dan **Preflighted Request**. Beberapa contoh request yang sudah kita pelajari di atas termasuk simple request.

Sedangkan mengenai preflighted request sendiri, mungkin pembaca secara tidak langsung juga pernah menerapkannya, terutama ketika bekerja di bagian front-end yang mengonsumsi data dari RESTful API yang servernya terpisah antara layer front end dan back end.

Ketika melakukan cross origin request dengan payload adalah JSON, atau request jenis lainnya, biasanya di developer tools -> network log muncul 2 kali request, request pertama method-nya `OPTIONS` dan request ke-2 adalah actual request.

Request ber-method `OPTIONS` tersebut disebut dengan **Preflight Request**. Request ini akan otomatis muncul ketika http request yang kita dispatch memenuhi kriteria preflighted request.

Tujuan dari preflight request adalah untuk mengecek apakah destinasi url mendukung CORS. Tiga buah informasi dikirimkan `Access-Control-Request-Method`, `Access-Control-Request-Headers`, dan `Origin`, dengan method adalah `OPTIONS`.

Berikut merupakan kriteria preflighted request.

- Method yang digunakan adalah salah satu dari method berikut:
  - `PUT`
  - `DELETE`
  - `CONNECT`
  - `OPTIONS`
  - `TRACE`
  - `PATCH`
- Terdapat header SELAIN yang otomatis di-set dalam http request. Contoh header untuk kriteria ini adalah `Authorization`, `X-CSRF-Token`, atau lainnya.
- Isi header `Content-Type` adalah SELAIN satu dari 3 berikut.
  - `application/x-www-form-urlencoded`
  - `multipart/form-data`
  - `text/plain`
- Ada event yang ter-registrasi dalam objek `XMLHttpRequestUpload` yang digunakan dalam request.
- Menggunakan objek `ReadableStream` dalam request.

Lebih detailnya mengenai simple dan preflighted request silakan baca <https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS>.

Pada kode yang kita tulis, terdapat pengecekan method `OPTIONS`. Pengecekan ini digunakan untuk mencegah eksekusi statement selanjutnya. Hal ini dikarenakan preflight request tidak membutuhkan kembalian data, tugas si dia hanya mengecek apakah cross origin request didukung atau tidak. Jadi pada handler, ketika method nya adalah `OPTIONS`, langsung saja intercept proses utamanya.

Header `Access-Control-Max-Age` diisi dengan data waktu, digunakan untuk menentukan seberapa lama informasi preflight request di-cache. Jika diisi dengan `-1` maka cache di-non-aktifkan.

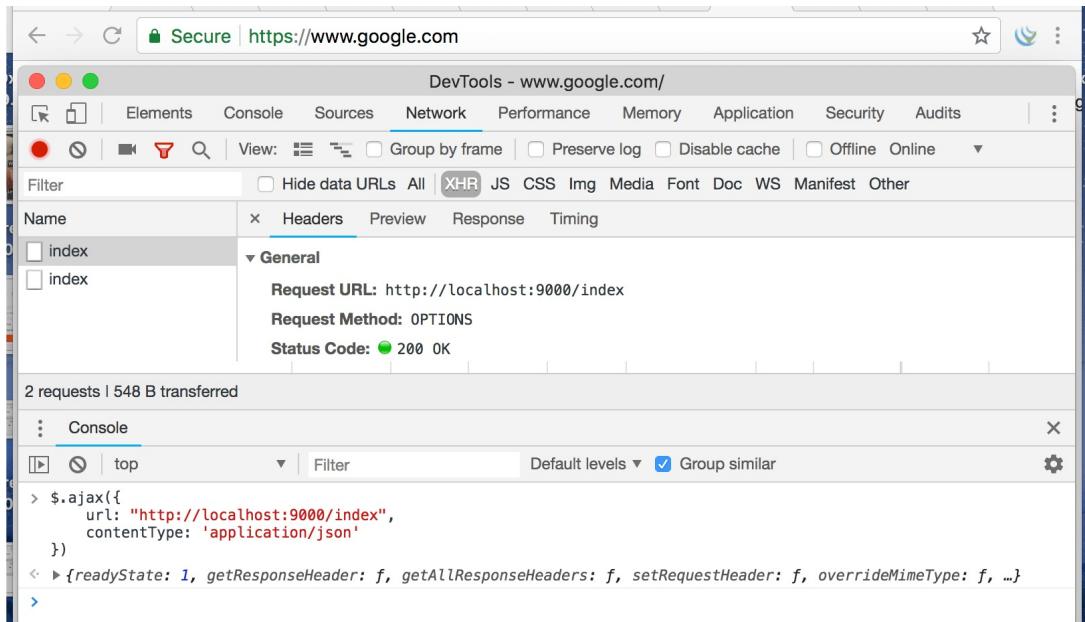
```
http.HandleFunc("/index", func(w http.ResponseWriter, r *http.Request) {
    // ...

    if r.Method == "OPTIONS" {
        w.Write([]byte("allowed"))
        return
    }

    // ...
})
```

## • Praktek

Langsung saja buka google.com lalu lakukan AJAX request yang memenuhi salah satu kriteria preflighted request, misalnya, gunakan header `Content-Type: application/json`.



Bisa dilihat pada screenshot, dua request muncul, yang pertama adalah preflight yang kedua adalah actual request.

## C.12.5. CORS Handling Menggunakan Golang CORS Library dan Echo

Pada bagian ini kita akan mengkombinasikan library CORS golang buatan Olivier Poitrey, dan Echo, untuk membuat back end yang mendukung cross origin request.

Pertama `go get` dulu library-nya.

```
$ go get https://github.com/rs/cors
```

Buat file baru, import library yang diperlukan lalu buat fungsi main.

```
package main

import (
    "github.com/labstack/echo"
    "github.com/rs/cors"
    "net/http"
)

func main() {
    e := echo.New()

    // ...

    e.GET("/index", func(c echo.Context) error {
        return c.String(http.StatusOK, "hello")
    })

    e.Logger.Fatal(e.Start(":9000"))
}
```

Siapkan objek `corsMiddleware`, cetak dari fungsi `cors.New()`. Pada parameter konfigurasi, isi spesifikasi CORS sesuai kebutuhan.

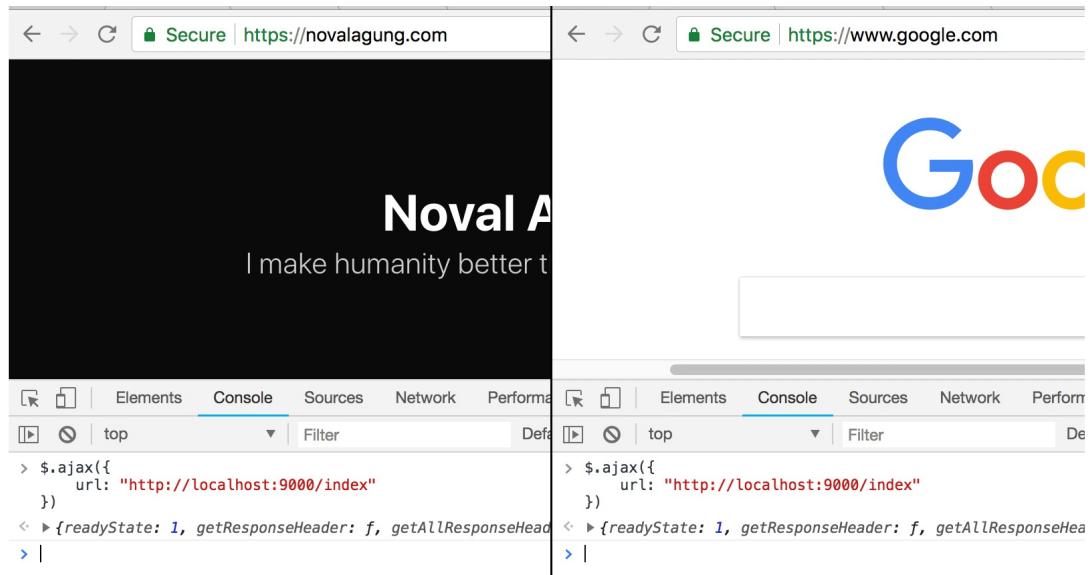
Gaya konfigurasi library ini menarik, mudah sekali untuk dipahami.

```
corsMiddleware := cors.New(cors.Options{
    AllowedOrigins: []string{"https://novalagung.com", "https://www.google.com"},
    AllowedMethods: []string{"OPTIONS", "GET", "POST", "PUT"},
    AllowedHeaders: []string{"Content-Type", "X-CSRF-Token"},
    Debug:           true,
})
e.Use(echo.WrapMiddleware(corsMiddleware.Handler))
```

Pada kode di atas, kita meng-allow dua buah origin. Sebelumnya sudah kita bahas bahwa kebanyakan browser tidak mendukung ini. Dengan menggunakan CORS library, hal itu bisa teratasi.

Sebenarnya mekanisme yang diterapkan oleh CORS library adalah meng-allow semua origin, lalu kemudian mem-filter sesuai dengan spesifikasi yang kita buat, lalu memodifikasi response header `Access-Control-Allow-Origin`-nya.

Jalankan aplikasi, coba test dari dua domain, <https://novalagung.com> dan <https://www.google.com>.



Berikut adalah list konfigurasi yang bisa dimanfaatkan dari library ini.

Key	Description
AllowedOrigins	list origin/domain yang diperbolehkan mengakses, gunakan * untuk allow all
AllowOriginFunc	callback untuk validasi origin. cocok digunakan untuk menge-set CORS header origin dengan ijin rumit
AllowedMethods	list HTTP method yang diperbolehkan untuk pengaksesan
AllowedHeaders	list header yang diperbolehkan untuk pengaksesan
ExposedHeaders	menentukan header mana saja yang di-expose ke consumer
AllowCredentials	enable/disable credentials
MaxAge	durasi cache preflight request
OptionsPassthrough	digunakan untuk menginstruksikan handler selanjutnya untuk memproses OPTIONS method

Debug

aktifkan properti ini pada stage development, agar banyak informasi log tambahan bisa muncul

---

- [CORS](#), by Olivier Poitrey, MIT license
- [Echo](#), by Vishal Rana (Lab Stack), MIT license

## C.13. CSRF

Pada bab ini kita akan belajar tentang serangan Cross-Site Request Forgery (CSRF) dan cara mengatasinya.

### C.13.1. Teori

Cross-Site Request Forgery atau CSRF adalah salah satu teknik hacking yang dilakukan dengan cara mengeksekusi perintah yang seharusnya tidak diizinkan, tetapi output yang dihasilkan sesuai dengan yang seharusnya. Contoh serangan jenis ini: mencoba untuk login lewat media selain web browser, seperti menggunakan CURL, menembak langsung endpoint login. Masih banyak contoh lainnya yang lebih ekstrim.

Ada beberapa cara untuk mencegah serangan ini, salah satunya adalah dengan memanfaatkan csrf token. Di setiap halaman yang ada form nya, csrf token di-generate. Pada saat submit form, csrf disisipkan di request, lalu di sisi back end dilakukan pengecekan apakah csrf yang dikirim valid atau tidak.

Csrf token sendiri merupakan sebuah random string yang di-generate setiap kali halaman form muncul. Biasanya di tiap POST request, token tersebut disisipkan sebagai header, atau form data, atau query string.

Lebih detailnya silakan merujuk ke [https://en.wikipedia.org/wiki/Cross-site\\_request\\_forgery](https://en.wikipedia.org/wiki/Cross-site_request_forgery).

## B.14.2. Praktek: Back End

Di golang, pencegahan CSRF bisa dilakukan dengan membuat middleware untuk pengecekan setiap request POST yang masuk. Cukup mudah sebenarnya, namun agar lebih mudah lagi kita akan gunakan salah satu middleware milik echo framework untuk belajar.

Di setiap halaman, jika di dalam html nya terdapat form, maka harus disisipkan token csrf. Token tersebut di-generate oleh middleware.

Di tiap POST request hasil dari form submit, token tersebut harus ikut dikirimkan. Proses validasi token sendiri di-handle oleh middleware.

Mari kita praktikan, siapkan projek baru. Buat file main, isi dengan kode berikut.

```
package main

import (
    "fmt"
    "github.com/labstack/echo"
    "github.com/labstack/echo/middleware"
    "html/template"
    "net/http"
)

type M map[string]interface{}

func main() {
    tmpl := template.Must(template.ParseGlob("./*.html"))

    e := echo.New()

    // ...

    e.Logger.Fatal(e.Start(":9000"))
}
```

Nantinya akan ada endpoint `/index`, isinya menampilkan html form. Objek `tmpl` kita gunakan untuk rendering form tersebut. API echo renderer tidak digunakan dalam bab ini.

Siapkan routing untuk `/index`, dan registrasikan middleware CSRF.

```
const CSRF_TOKEN_HEADER = "X-Csrf-Token"
const CSRF_KEY = "csrf_token"

e.Use(middleware.CSRFWithConfig(middleware.CSRFConfig{
    TokenLookup: "header:" + CSRF_TOKEN_HEADER,
    ContextKey:  CSRF_KEY,
}))

e.GET("/index", func(c echo.Context) error {
    data := make(M)
    data[CSRF_KEY] = c.Get(CSRF_KEY)
    return tmpl.Execute(c.Response(), data)
})
```

Objek middleware CSRF dibuat lewat statement `middleware.CSRF()`, konfigurasi default digunakan. Atau bisa juga dibuat dengan disertakan konfigurasi custom, lewat `middleware.CSRFWithConfig()` seperti pada kode di atas.

Property `ContextKey` digunakan untuk mengakses token csrf yang tersimpan di `echo.Context`, pembuatan token sendiri terjadi pada saat ada http request GET masuk.

Property tersebut kita isi dengan konstanta `CSRF_KEY`, maka dalam pengambilan token cukup panggil `c.Get(CSRF_KEY)`. Token kemudian disisipkan sebagai data pada saat rendering `view.html`.

Property `TokenLookup` adalah acuan di bagian mana informasi csrf disisipkan dalam objek request, apakah dari header, query string, atau form data. Ini penting karena dibutuhkan oleh middleware yang bersangkutan untuk memvalidasi token tersebut. Bisa dilihat, kita memilih `header:X-Csrf-Token`, artinya csrf token dalam request akan disisipkan dalam header dengan key adalah `x-csrf-token`.

Isi value `TokenLookup` dengan `"form:<name>"` jika token disispkan dalam form data request, dan `"query:<name>"` jika token disispkan dalam query string.

Selanjutnya siapkan satu endpoint lagi, yaitu `/sayhello`, endpoint ini nantinya menjadi tujuan request yang di-dispatch dari event submit form.

```
e.POST("/sayhello", func(c echo.Context) error {
    data := make(M)
    if err := c.Bind(&data); err != nil {
        return err
    }

    message := fmt.Sprintf("hello %s", data["name"])
    return c.JSON(http.StatusOK, message)
})
```

Pada handler endpoint `/sayhello` tidak ada pengecekan token csrf, karena sudah ditangani secara implisit oleh middleware.

## B.14.3. Front End

Buat `view.html`, lalu isi kode berikut.

```
<!DOCTYPE html>
<html>
<head>
```

```

<title></title>
</head>
<body>
    <form id="form" action="/sayhello" method="POST">
        <div>
            <label>Name</label>
            <input type="text" name="name" placeholder="Type your name here">
        </div>
        <div>
            <label>Gender</label>
            <select name="gender">
                <option value="">Select one</option>
                <option value="male">Male</option>
                <option value="female">Female</option>
            </select>
        </div>
        <div>
            <input type="hidden" name="csrf_token" value="{{ .csrf_token }}">
            <button type="submit">Submit</button>
        </div>
    </form>

    <script src="https://code.jquery.com/jquery-3.3.1.min.js"></script>

    <script type="text/javascript">
        // JS code here ...
    </script>
</body>
</html>

```

Bisa dilihat, sebuah form disiapkan dengan isi 2 inputan dan satu buah tombol submit.

Sebenarnya ada tiga inputan, namun yang satu adalah hidden. Inputan tersebut berisi nilai `csrf_token` yang disisipkan dari back end.

Pada saat tombol submit di-klik, token tersebut harus disisipkan dalam AJAX request yang mengarah ke `/sayhello`.

Sekarang buat script JS-nya. Siapkan sebuah event listener `submit` untuk element `form`, isinya adalah AJAX. Ambil informasi inputan nama dan gender, jadikan sebagai payload AJAX tersebut.

```

$(function () {
    $('form').on('submit', function (e) {
        e.preventDefault()

        var self = $(this)

        var formData = {
            name: self.find('[name="name"]').val(),
            gender: self.find('[name="gender"]').val(),
        }

        var url = self.attr('action')
        var method = self.attr('method')
        var payload = JSON.stringify(formData)

        $.ajax({
            url: url,
            type: method,
            contentType: 'application/json',
            data: payload,
            beforeSend: function(req) {
                var csrfToken = self.find('[name=csrf_token]').val()
                req.setRequestHeader("X-Csrf-Token", csrfToken)
            },
        }).then(function (res) {
            alert(res)
        })
    })
})

```

```

        }).catch(function (err) {
            alert('ERROR: ' + err.responseText)
            console.log('err', err)
        })
    })
})
}

```

Tambahkan header `x-csrf-Token` di AJAX request seperti pada kode di atas, isinya diambil dari inputan hidden `csrf_token`. Nama header sendiri menggunakan `x-Csrf-Token`.

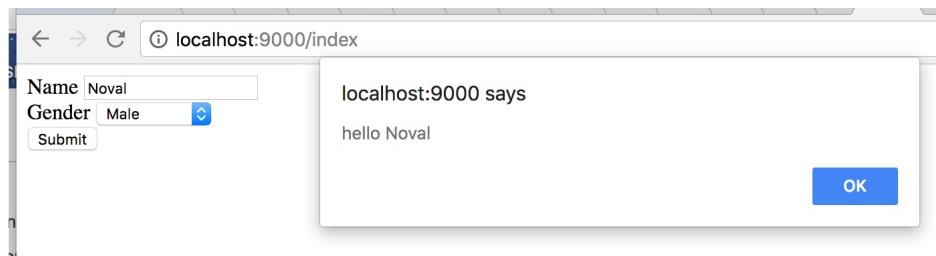
Karena di konfigurasi middleware csrf di back end `TokenLookup` adalah `header:x-Csrf-Token`, maka header dengan nama `x-Csrf-Token` dipilih.

## C.13.4. Testing

Sekarang jalankan aplikasi lalu akses `/index` untuk mengetes hasilnya. Silakan melakukan skenario testing berikut.

1. Buka laman `/index`, form akan muncul.
2. Pada saat rendering output `/index`, disisipkan juga token csrf yang di-generate oleh middleware pada saat endpoint ini diakses.
3. OK, sekarang laman form sudah muncul.
4. Isi inputan form.
5. Klik tombol submit.
6. Di event submit tersebut, sebuah AJAX dipersiapkan dengan tujuan adalah `/sayhello`.
7. Di dalam AJAX ini, token csrf yang sebelumnya disisipkan, diambil lalu ditempelkan ke AJAX.
8. AJAX di-dispatch ke `/sayhello`.
9. Sebelum benar-benar diterima oleh handler endpoint, middleware secara otomatis melakukan pengecekan atas token csrf yang disisipkan.
10. Jika pengecekan sukses, maka request diteruskan ke tujuan.
11. Jika tidak sukses, error message dikembalikan.
12. Pengecekan adalah sukses, alert message muncul.

Hasilnya:

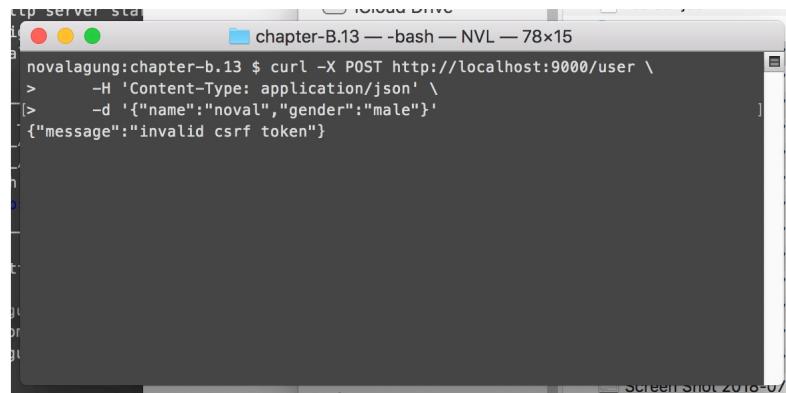


Coba tembak langsung endpoint nya lewat CURL.

```

$ curl -X POST http://localhost:9000/user \
-H 'Content-Type: application/json' \
-d '{"name":"noval","gender":"male"}'

```



```
curl -X POST http://localhost:9000/user \
-H 'Content-Type: application/json' \
-d '{"name":"noval","gender":"male"}'
{"message":"invalid csrf token"}
```

Hasilnya error, karena token csrf tidak di-sisipkan.

Lewat teknik pencegahan ini, bukan berarti serangan CSRF tidak bisa dilakukan, si hacker masih bisa menembak endpoint secara paksa lewat CURL, hanya saja membutuhkan usaha ekstra jika ingin sukses.

- 
- [Echo](#), by Vishal Rana (Lab Stack), MIT license

## C.14. Secure Middleware

Pada bab ini kita akan belajar menggunakan library `secure` untuk meningkatkan keamanan aplikasi web.

### C.14.1. Keamanan Web Server

Jika berbicara mengenai keamanan aplikasi web, sangat luas sebenarnya cakupannya, ada banyak hal yang perlu diperhatian dan disiapkan. Mungkin tiga diantaranya sudah kita pelajari sebelumnya, yaitu penerapan Secure Cookie, CORS, dan CSRF.

Selain 3 topik tersebut masih terdapat banyak lagi. Beruntungnya ada library `secure`. Sesuai tagline-nya, secure library digunakan untuk membantu mengatasi beberapa masalah keamanan aplikasi.

Secure library merupakan middleware, penggunaannya sama seperti middleware pada umumnya.

### C.15.2. Praktek

Mari langsung kita praktikan. Buat folder projek baru. Di file main tulis kode berikut. Sebuah aplikasi dibuat, isinya satu buah rute `/index` yang bisa diakses dari mana saja.

```
package main

import (
    "net/http"
    "github.com/labstack/echo"
)

func main() {
    e := echo.New()

    e.GET("/index", func(c echo.Context) error {
        c.Response().Header().Set("Access-Control-Allow-Origin", "*")

        return c.String(http.StatusOK, "Hello")
    })

    e.Logger.Fatal(e.StartTLS(":9000", "server.crt", "server.key"))
}
```

Perlu diketahui, aplikasi di atas di-start dengan SSL/TLS enabled. Dua buah file dibutuhkan, yaitu file certificate `server.crt` dan file private key `server.key`. Silakan unduh kedua file tersebut dari source code di [github, bab B14](#). Pada bab [B.22 HTTPS/TLS Web Server](#) nantinya akan kita pelajari lebih lanjut mengenai cara generate kedua file di atas hingga cara penggunannya.

Kembali ke pembahasan, sekarang tambahkan secure middleware. Import package-nya, buat instance middleware, lalu registrasikan ke echo.

```
import (
    // ...
    "github.com/unrolled/secure"
)

func main() {
    // ...
```

```

secureMiddleware := secure.New(secure.Options{
    AllowedHosts:          []string{"localhost:9000", "www.google.com"},
    FrameDeny:              true,
    CustomFrameOptionsValue: "SAMEORIGIN",
    ContentTypeNosniff:     true,
    BrowserXssFilter:      true,
})

e.Use(echo.WrapMiddleware(secureMiddleware.Handler))

// ...
}

```

Pembuatan objek secure middleware dilakukan menggunakan `secure.New()` dengan isi parameter adalah konfigurasi. Bisa dilihat ada 5 buah property konfigurasi di-set. Berikut merupakan penjelasan tiap-tiap property tersebut.

### • Konfigurasi `AllowedHosts`

```
AllowedHosts: []string{"localhost:9000", "www.google.com"}
```

Host yang diperbolehkan mengakses web server ditentukan hanya 2, yaitu localhost:9000 yang merupakan web server itu sendiri, dan google.com. Silakan coba mengakses aplikasi kita ini menggunakan AJAX lewat google.com dan domainnya lainnya untuk mengetes apakah fungsionalitas nya berjalan.

### • Konfigurasi `FrameDeny`

```
FrameDeny: true
```

Secara default sebuah aplikasi web adalah bisa di-load di dalam iframe yang berada host nya berbeda. Misalnya di salah satu laman web www.kalipare.com ada iframe yang atribut src nya berisi www.novalagung.com, hal seperti ini diperbolehkan.

Perijinan apakah website boleh di-load lewat iframe atau tidak, dikontrol lewat header [X-Frame-Options](#).

Di library secure, untuk men-disable ijin akses aplikasi dari dalam iframe, bisa dilakukan cukup dengan mengeset proerty `FrameDeny` dengan nilai `true`.

Untuk mengetes, silakan buat aplikasi web terpisah yang mer-render sebuah view. Dalam view tersebut siapkan satu buah iframe yang mengarah ke `https://localhost:9000/index`.

### • Konfigurasi `CustomFrameOptionsValue`

```
CustomFrameOptionsValue: "SAMEORIGIN"
```

Jika `FrameDeny` di-set sebagai `true`, maka semua host (termasuk aplikasi itu sendiri) tidak akan bisa me-load url lewat iframe.

Dengan menambahkan satu buah property lagi yaitu `CustomFrameOptionsValue: "SAMEORIGIN"` maka ijin pengaksesan url lewat iframe menjadi eksklusif hanya untuk aplikasi sendiri.

Untuk mengetes, buat rute baru yang me-render sebuah view. Dalam view tersebut siapkan satu buah iframe yang mengarah ke `/index`.

### • Konfigurasi `ContentTypeNosniff`

```
ContentTypeNosniff: true
```

Property `ContentTypeNosniff: true` digunakan untuk disable MIME-sniffing yang dilakukan oleh browser IE. Lebih jelasnya silakan baca [X-Content-Type-Options](#).

- **Konfigurasi BrowserXssFilter**

```
BrowserXssFilter: true
```

Property di atas digunakan untuk mengaktifkan header [X-XSS-Protection](#), dengan isi header adalah `1; mode=block`.

### C.15.3. Property Library Secure

Selain 5 property yang kita telah pelajari di atas, masih ada banyak lagi konfigurasi yang bisa digunakan.

- AllowedHosts
- HostsProxyHeaders
- SSLRedirect
- SSLTemporaryRedirect
- SSLHost
- SSLHostFunc
- SSLProxyHeaders
- STSSeconds
- STSIncludeSubdomains
- STSPreload
- ForceSTSHeader
- FrameDeny
- CustomFrameOptionsValue
- ContentTypeNosniff
- BrowserXssFilter
- CustomBrowserXssValue
- ContentSecurityPolicy
- PublicKey
- ReferrerPolicy

Lebih mendetailnya silakan langsung cek halaman official library secure di <https://github.com/unrolled/secure>.

- 
- [Secure](#), by Cory Jacobsen, MIT license
  - [Echo](#), by Vishal Rana (Lab Stack), MIT license

## C.15. HTTP Gzip Compression (gziphandler)

Pada bab ini kita akan mempelajari penerapan HTTP Compression, dengan encoding adalah Gzip, dalam aplikasi web golang.

### C.15.1. Teori

HTTP Compression adalah teknik kompresi data pada HTTP response, agar ukuran/size output menjadi lebih kecil dan response time lebih cepat.

Pada saat sebuah endpoint diakses, di header request akan ada header `Accept-Encoding` yang disisipkan oleh browser secara otomatis.

```
GET /hello HTTP/1.1
Host: localhost:9000
Accept-Encoding: gzip, deflate
```

Jika isinya adalah `gzip` atau `deflate`, berarti browser siap dan support untuk menerima response yang di-compress dari back end.

Deflate adalah algoritma kompresi untuk data lossless. Gzip adalah salah satu teknik kompresi data yang menerapkan algoritma deflate.

Di sisi back end sendiri, jika memang output di-compress, maka response header `Content-Encoding: gzip` perlu disisipkan.

```
Content-Encoding: gzip
```

Jika di sebuah request tidak ada header `Accept-Encoding: gzip`, tetapi response back end tetap di-compress, maka akan muncul error di browser `ERR_CONTENT_DECODING_FAILED`.

### C.15.2. Praktek

Golang menyediakan package `compress/gzip`. Dengan memanfaatkan API yang tersedia dalam package tersebut, kompresi data pada HTTP response bisa dilakukan.

Namun pada bab ini kita tidak memakainya, melainkan menggunakan salah satu library middleware gzip compression yang cukup terkenal, [gziphandler](#).

Mari kita praktikan. Siapkan folder projek baru, siapkan satu buah route `/image`. Dalam handler route tersebut terdapat proses pembacaan isi file gambar `sample.png`, untuk kemudian dijadikan sebagai output data response. Gunakan file gambar apa saja untuk keperluan testing.

Tujuan dari aplikasi ini untuk melihat seberapa besar response size dan lama response time-nya. Nantinya akan kita bandingkan dengan hasil test di aplikasi yang menerapkan http gzip compression.

```
package main

import (
    "io"
    "net/http"
    "os"
```

```

)
func main() {
    mux := new(http.ServeMux)

    mux.HandleFunc("/image", func(w http.ResponseWriter, r *http.Request) {
        f, err := os.Open("sample.png")
        if f != nil {
            defer f.Close()
        }
        if err != nil {
            http.Error(w, err.Error(), http.StatusInternalServerError)
            return
        }

        _, err = io.Copy(w, f)
        if err != nil {
            http.Error(w, err.Error(), http.StatusInternalServerError)
        }
    })
}

server := new(http.Server)
server.Addr = ":9000"
server.Handler = mux

server.ListenAndServe()
}

```

Jalankan aplikasi lalu test hasilnya.

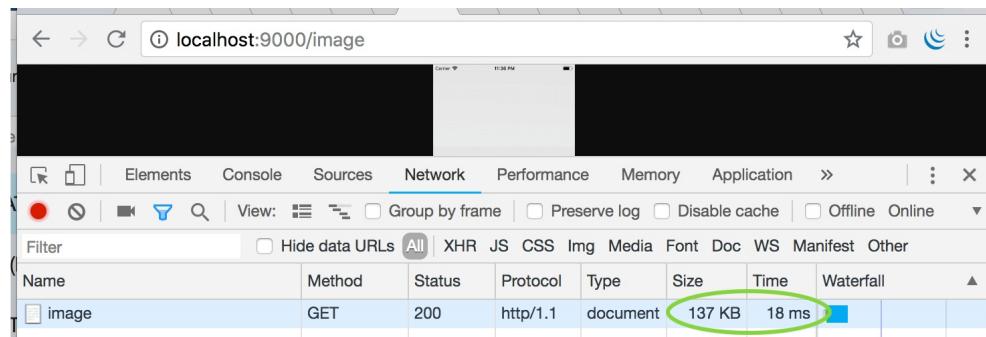


Image size adalah 137 KB, dan response time adalah 18 ms.

Selanjutnya kita akan coba menerapkan middleware gzip pada program kecil di atas. Pertama unduh dependency gziphandler terlebih dahulu menggunakan `go get`.

```
$ go get -u github.com/NYTimes/gziphandler
```

Import library yang sudah ter-unduh pada file main, lalu bungkus multiplexer `mux` menggunakan `gziphandler.GzipHandler()`, `mux` tersebut dimasukan ke property `server.Handler`.

```

import (
    // ...
    "github.com/NYTimes/gziphandler"
)

func main() {
    // ...

    server.Handler = gziphandler.GzipHandler(mux)

    // ...
}

```

Jalankan ulang aplikasi, lihat perbandingannya.

Name	Method	Status	Protocol	Type	Size	Time	Waterfall
image	GET	200	http/1.1	document	129 KB	16 ms	

Perbedannya size nya mungkin tidak terlihat signifikan, karena memang gambaranya berukuran kecil, jumlahnya cuma satu asset, dan pengaksesannya di localhost. Untuk aplikasi yang sudah published di internet, dan diakses dari komputer lokal, pasti akan terasa jauh lebih cepat dan ringan.

### C.15.3. Gzip Compression di Echo

Penerapan http gzip compression di echo framework bisa dengan menggunakan middleware gziphandler di atas. Atau bisa juga menggunakan middleware gzip milik echo. Berikut merupakan contoh pemanfaatan echo middleware gzip.

```
e := echo.New()

e.Use(middleware.Gzip())

e.GET("/image", func(c echo.Context) error {
    f, err := os.Open("sample.png")
    if err != nil {
        return err
    }

    _, err = io.Copy(c.Response(), f)
    if err != nil {
        return err
    }

    return nil
})

e.Logger.Fatal(e.Start(":9000"))
```

- [Gzip Handler](#), by The New York Times team, Apache-2.0 license

- [Echo](#), by Vishal Rana (Lab Stack), MIT license

## C.16. Send Mail ( net/smtp , Gomail v2)

Pada bab ini kita akan belajar cara mengirim email dari aplikasi golang, menggunakan dua cara berikut.

1. Dengan memanfaatkan package `net/smtp` .
2. Menggunakan Library [gomail](#).

### C.16.1. Kirim Email Menggunakan `net/smtp`

Golang menyediakan package `net/smtp` , isinya banyak API untuk berkomunikasi via protokol SMTP. Lewat package ini kita bisa melakukan operasi kirim email.

Sebuah akun email diperlukan dalam mengirim email, silakan gunakan provider email apa saja. Pada bab ini kita gunakan Google Mail (gmail), jadi siapkan satu buah akun gmail untuk keperluan testing.

Mari kita praktikan. Buat folder projek baru, salin kode berikut.

```
package main

import (
    "fmt"
    "log"
    "net/smtp"
    "strings"
)

const CONFIG_SMTP_HOST = "smtp.gmail.com"
const CONFIG_SMTP_PORT = 587
const CONFIG_EMAIL = "emailanda@gmail.com"
const CONFIG_PASSWORD = "passwordemailanda"

func main() {
    to := []string{"recipient1@gmail.com", "emaillain@gmail.com"}
    cc := []string{"tralalala@gmail.com"}
    subject := "Test mail"
    message := "Hello"

    err := sendMail(to, cc, subject, message)
    if err != nil {
        log.Fatal(err.Error())
    }

    log.Println("Mail sent!")
}
```

Dalam implementasinya, untuk bisa mengirim email, dibutuhkan mail server. Karena kita menggunakan email google, maka mail server milik google digunakan.

Pada kode di atas, konstanta dengan prefix `CONFIG_` adalah konfigurasi yang diperlukan untuk terhubung dengan mail server. Sesuaikan `CONFIG_EMAIL` dan `CONFIG_PASSWORD` dengan akun yang digunakan.

Di dalam fungsi main bisa dilihat, fungsi `sendMail()` dipanggil untuk mengirim email, dengan empat buah parameter disisipkan.

- Parameter `to` , adalah tujuan email.
- Parameter `cc` , adalah cc tujuan.
- Parameter `subject` , adalah subjek email.
- Parameter `message` , adalah body email.

Email pada konstanta `CONFIG_EMAIL` menjadi sender email.

OK, selanjutnya buat fungsi `sendMail()` berikut.

```
func sendMail(to []string, cc []string, subject, message string) error {
    body := "From: " + CONFIG_EMAIL + "\n" +
        "To: " + strings.Join(to, ",") + "\n" +
        "Cc: " + strings.Join(cc, ",") + "\n" +
        "Subject: " + subject + "\n\n" +
        message

    auth := smtp.PlainAuth("", CONFIG_EMAIL, CONFIG_PASSWORD, CONFIG_SMTP_HOST)
    smtpAddr := fmt.Sprintf("%s:%d", CONFIG_SMTP_HOST, CONFIG_SMTP_PORT)

    err := smtp.SendMail(smtpAddr, auth, CONFIG_EMAIL, append(to, cc...), []byte(body))
    if err != nil {
        return err
    }

    return nil
}
```

Fungsi `sendMail()` digunakan untuk mengirim email. Empat data yang disisipkan pada fungsi tersebut dijadikan satu dalam format tertentu, lalu disimpan ke variabel `body`.

Statement yang ditampung oleh `body` akan menghasilkan string berikut (formatnya adalah baku).

```
From: recipient1@gmail.com, emaillain@gmail.com
Cc: tralalala@gmail.com
Subject: Test mail

Hello
```

Pengiriman email dilakukan lewat `smtp.SendMail()`. Dalam pemanggilannya 5 buah parameter disisipkan, berikut adalah penjelasan masing-masing parameter.

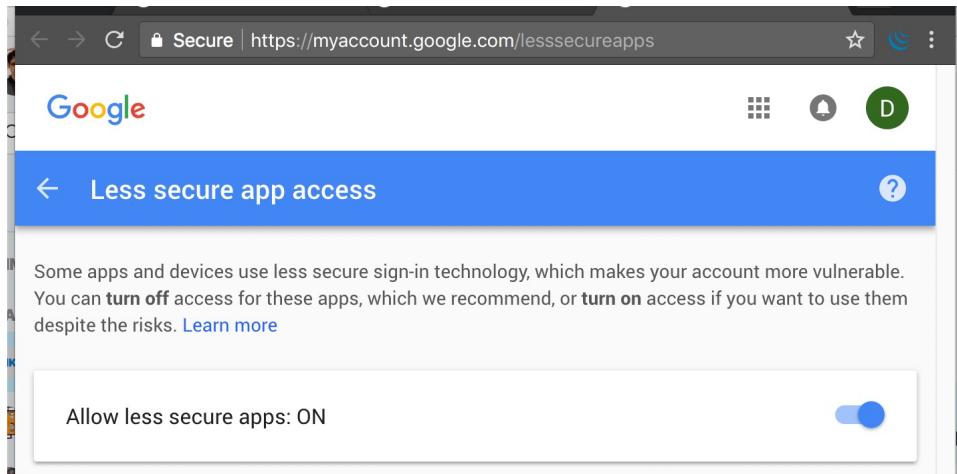
- Parameter ke-1, `smtpAddr`, merupakan kombinasi host dan port mail server.
- Parameter ke-2, `auth`, menampung credentials untuk keperluan otentikasi ke mail server. Objek ini dicetak lewat `smtp.PlainAuth()`.
- Parameter ke-3, `CONFIG_EMAIL`, adalah alamat email yang digunakan untuk mengirim email.
- Parameter ke-4, Isinya adalah semua email tujuan, termasuk `cc`.
- Parameter ke-5, isinya `body` email.

Jalankan aplikasi. Lihat di console, error muncul.

```
[novalagung:chapter-b.16 $ go run main.go
2018/08/02 11:19:35 --- 534 5.7.14 <https://accounts.google.com/signin/continue?
sarp=1&sccc=1&pl...>h
5.7.14 10piIQsV3hwu2lK...>DqWfM2iXzuzByL3ZIeN0p06csSYKB7ia-2gQ9uxhax9yVN
5.7.14 lyfelUxvNg-JAPPkjWUfogcffPxq5GwMn...>T03H5voVY...>u1-uLwp1
5.7.14 jD4lfif2'>T03H5voVY...>Jic0LXUPCUYXqRAk59mTBssTGz_kbnchg_Ag1rqN5ffFBu
5.7.14 jKl62HDaLRgAk1T7gVST_E0...>T03H5voVY...>2P_j5Rt09> Please log in via
5.7.14 your web browser and then try again.
5.7.14 Learn more at
5.7.14 https://support.google.com/mail/answer/78754 f6-.....66pgf.52 - gsmtp]
```

Error di atas hanya muncul pada pengiriman email menggunakan akun google mail. Untuk alasan keamanan, google men-disable akun gmail untuk digunakan mengirim email lewat kode program.

Aktifkan fasilitas **less secure apps** untuk meng-enable-nya. Login ke gmail masing-masing, kemudian buka link <https://myaccount.google.com/lesssecureapps>, lalu klik tombol toggle agar menjadi **OFF**.



Jalankan ulang aplikasi, email terkirim. Lihat di inbox email tujuan pengiriman untuk mengecek hasilnya.



## C.16.2. Kirim Email Menggunakan Gomail v2

Dengan library [gomail](#), pengiriman email bisa dilakukan dengan mudah. Beberapa operasi seperti membuat email dalam bentuk html, menambahkan attachment, menambahkan bcc, bisa dilakukan dengan mudah lewat gomail.

Mari langsung kita praktikan. Unduh terlebih dahulu library-nya.

```
$ go get -u gopkg.in/gomail.v2
```

Lalu tulis kode berikut.

```
package main

import (
    "gopkg.in/gomail.v2"
    "log"
)

const CONFIG_SMTP_HOST = "smtp.gmail.com"
const CONFIG_SMTP_PORT = 587
const CONFIG_EMAIL = "emailanda@gmail.com"
const CONFIG_PASSWORD = "passwordemailanda"

func main() {
    mailer := gomail.NewMessage()
    mailer.SetHeader("From", CONFIG_EMAIL)
    mailer.SetHeader("To", "recipient1@gmail.com", "emaillain@gmail.com")
    mailer.SetAddressHeader("Cc", "tralalala@gmail.com", "Tra La La La")
    mailer.SetHeader("Subject", "Test mail")
    mailer.SetBody("text/html", "Hello, <b>have a nice day</b>")
    mailer.Attach("./sample.png")

    dialer := gomail.NewDialer(
```

```

    CONFIG_SMTP_HOST,
    CONFIG_SMTP_PORT,
    CONFIG_EMAIL,
    CONFIG_PASSWORD,
)

err := dialer.DialAndSend(mail)
if err != nil {
    log.Fatal(err.Error())
}

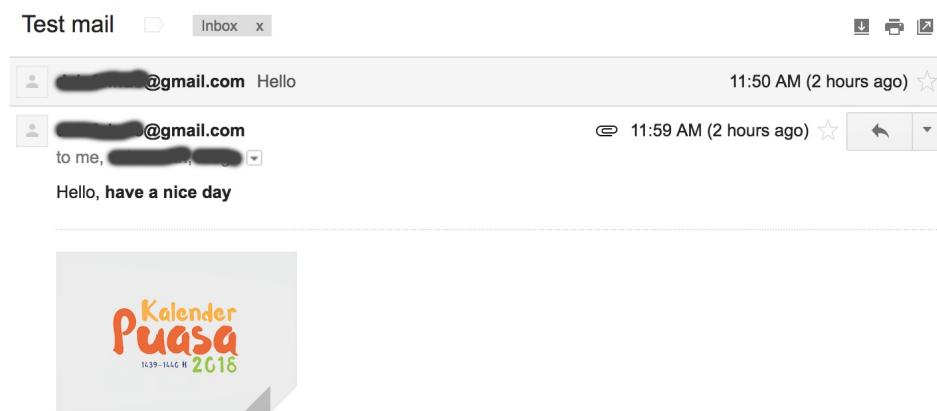
log.Println("Mail sent!")
}

```

Siapkan satu buah image bernama `sample.png`, simpan di dalam folder yang sama dengan file main. Untuk meng-attach file ke dalam email, gunakan method `.Attach()` milik `*gomail.Message`.

Pada contoh kali ini email isinya adalah HTML. Gunakan MIME html pada parameter pertama `.SetBody()` untuk mengaktifkan mode html email.

Jalankan aplikasi, lalu cek hasilnya email yang dikirim di inbox.



- [Gomail v2](#), by Alexandre Cesaro, MIT license

## BC17. Read & Write Excel XLSX File (Excelize)

Dalam pengembangan aplikasi web, di bagian reporting, tidak jarang kita akan berususan dengan file excel. Biasanya di tiap report diharuskan ada fasilitas untuk unduh data ke bentuk excel ataupun pdf.

Pada bab ini kita akan belajar tentang pengolahan file excel menggunakan [excelize](#).

Dokumentasi lengkap mengenai excelize bisa dilihat di <https://xuri.me/excelize/en>. Silakan `go get` untuk mengunduh library ini.

```
$ go get github.com/360EntSecGroup-Skylar/excelize
```

### C.17.1. Membuat File Excel .xlsx

Pembahasan akan dilakukan dengan langsung praktik, dengan skenario: sebuah dummy data bertipe `[]M` disiapkan, data tersebut kemudian ditulis ke dalam excel.

Buat projek baru, buat file main, import excelize dan siapkan dummy data-nya.

```
package main

import (
    "fmt"
    "github.com/360EntSecGroup-Skylar/excelize"
    "log"
)

type M map[string]interface{}

var data = []M{
    M{"Name": "Noval", "Gender": "male", "Age": 18},
    M{"Name": "Nabila", "Gender": "female", "Age": 12},
    M{"Name": "Yasa", "Gender": "male", "Age": 11},
}

func main() {
    // magic here
}
```

Di fungsi `main()` buat objek excel baru, menggunakan `excelize.NewFile()`. Secara default, objek excel memiliki satu buah sheet dengan nama `Sheet1`.

```
xlsx := excelize.NewFile()

sheet1Name := "Sheet One"
xlsx.SetSheetName(xlsx.GetSheetName(1), sheet1Name)
```

Sheet `Sheet1` kita ubah namanya menjadi `Sheet One` lewat statement `xlsx.SetSheetName()`. Perlu diperhatikan index sheet dimulai dari 1, bukan 0.

Siapkan cell header menggunakan kode berikut.

```
xlsx.SetCellValue(sheet1Name, "A1", "Name")
xlsx.SetCellValue(sheet1Name, "B1", "Gender")
xlsx.SetCellValue(sheet1Name, "C1", "Age")
```

```
err := xlsx.AutoFilter(sheet1Name, "A1", "C1", "")
if err != nil {
    log.Fatal("ERROR", err.Error())
}
```

Penulisan cell dilakukan lewat method `.SetCellValue()` milik objek excel. Pemanggilannya membutuhkan 3 buah parameter untuk disisipkan.

1. Parameter ke-1, sheet name.
2. Parameter ke-2, lokasi cell.
3. Parameter ke-3, nilai/text/isi cell.

Pada kode di atas, cell `A1`, `B1`, dan `C1` disiapkan dan diaktifkan filter didalamnya. Cara mengeset filter pada cell sendiri dilakukan lewat method `.AutoFilter()`. Tentukan range lokasi cell sebagai parameter.

Statement `xlsx.AutoFilter(sheet1Name, "A1", "C1", "")` artinya filter diaktifkan pada sheet `sheet1Name` mulai cell `A1` hingga `C1`.

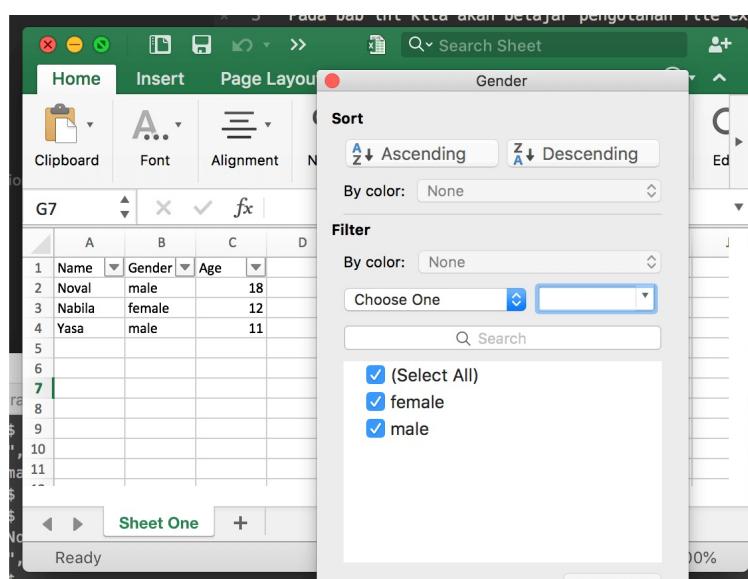
Lalu lakukan perulangan pada `data`. Tulis tiap map item sebagai cell berurutan per row setelah cell header.

```
for i, each := range data {
    xlsx.SetCellValue(sheet1Name, fmt.Sprintf("A%d", i+2), each["Name"])
    xlsx.SetCellValue(sheet1Name, fmt.Sprintf("B%d", i+2), each["Gender"])
    xlsx.SetCellValue(sheet1Name, fmt.Sprintf("C%d", i+2), each["Age"])
}
```

Terakhir simpan objek excel sebagai file fisik. Gunakan `.SaveAs()`, isi parameter dengan path lokasi excel akan disimpan.

```
err = xlsx.SaveAs("./file1.xlsx")
if err != nil {
    fmt.Println(err)
}
```

Jalankan aplikasi, sebuah file bernama `file1.xlsx` akan muncul. Buka file tersebut lihat isinya. Data tersimpan sesuai ekspektasi. Fasilitas filter pada cell header juga aktif.



## C.17.2. Pembuatan Sheet, Merge Cell, dan Cell Styles

Manajemen sheet menggunakan excelize cukup mudah. Pembuatan sheet dilakukan lewat `xlsx.NewSheet()`. Mari langsung kita praktikan.

Hapus baris kode mulai statement `xlsx.SaveAs()` kebawah. Lalu tambahkan kode berikut.

```
sheet2Name := "Sheet two"
sheetIndex := xlsx.NewSheet(sheet2Name)
xlsx.SetActiveSheet(sheetIndex)
```

Statement `xlsx.SetActiveSheet()` digunakan untuk menge-set sheet yang aktif ketika file pertama kali dibuka.

Parameter yang dibutuhkan adalah index sheet.

Pada sheet baru, `Sheet two`, tambahkan sebuah text `Hello` pada sheet `A1`, lalu merge cell dengan cell di sebelah kanannya.

```
xlsx.SetCellValue(sheet2Name, "A1", "Hello")
xlsx.MergeCell(sheet2Name, "A1", "B1")
```

Tambahkan juga style pada cell tersebut. Buat style baru lewat `xlsx.NewStyle()`, sisipkan konfigurasi style sebagai parameter.

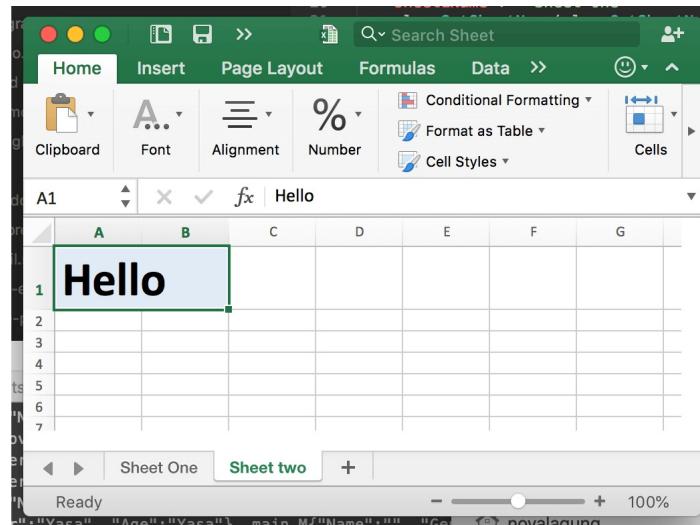
```
style, err := xlsx.NewStyle(`{
    "font": {
        "bold": true,
        "size": 36
    },
    "fill": {
        "type": "pattern",
        "color": ["#E0EBF5"],
        "pattern": 1
    }
}`)
if err != nil {
    log.Fatal("ERROR", err.Error())
}
xlsx.SetCellStyle(sheet2Name, "A1", "A1", style)

err = xlsx.SaveAs("./file2.xlsx")
if err != nil {
    fmt.Println(err)
}
```

Di excelize, style merupakan objek terpisah. Kita bisa mengaplikasikan style tersebut ke cell mana saja menggunakan `xlsx.SetCellStyle()`.

Silakan merujuk ke <https://xuri.me/excelize/en/cell.html#SetCellStyle> untuk pembahasan yang lebih detail mengenai cell style.

Sekarang jalankan aplikasi, lalu coba buka file `file2.xlsx`.



### C.17.3. Membaca File Excel .xlsx

Di excelize, objek excel bisa didapat lewat dua cara.

- Dengan membuat objek excel baru menggunakan `excelize.NewFile()`.
- Atau dengan membaca file excel lewat `excelize.OpenFile()`.

Dari objek excel, operasi baca dan tulis bisa dilakukan. Berikut merupakan contoh cara membaca file excel yang sudah kita buat, `file1.xlsx`.

```
xlsx, err := excelize.OpenFile("./file1.xlsx")
if err != nil {
    log.Fatal("ERROR", err.Error())
}

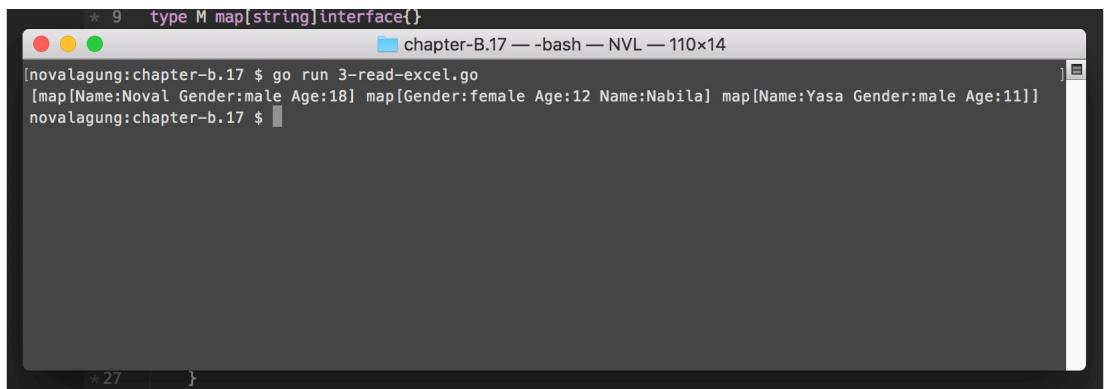
sheet1Name := "Sheet One"

rows := make([]M, 0)
for i := 2; i < 5; i++ {
    row := M{
        "Name":    xlsx.GetCellValue(sheet1Name, fmt.Sprintf("A%d", i)),
        "Gender":  xlsx.GetCellValue(sheet1Name, fmt.Sprintf("B%d", i)),
        "Age":     xlsx.GetCellValue(sheet1Name, fmt.Sprintf("C%d", i)),
    }
    rows = append(rows, row)
}

fmt.Printf("%v \n", rows)
```

Pada kode di atas, data tiap cell diambil lalu ditampung ke slice `M`. Gunakan `xlsx.GetCellValue()` untuk mengambil data cell.

Jalankan aplikasi untuk mengecek hasilnya.



A screenshot of a terminal window titled "chapter-B.17 — bash — NVL — 110x14". The window shows the following command and its output:

```
* 9 type M map[string]interface{}  
[novalagung:chapter-b.17 $ go run 3-read-excel.go  
[map[Name:Noval Gender:male Age:18] map[Gender:female Age:12 Name:Nabila] map[Name:Yasa Gender:male Age:11]]  
novalagung:chapter-b.17 $
```

- 
- [Excelize](#), by 360 Enterprise Security Group Team, BSD 3 Clause license

## C.18. Write PDF File (gofpdf)

Reporting pada aplikasi web, selain ke bentuk file excel biasanya ke bentuk file pdf. Pada bab ini kita akan mempelajari cara membuat file pdf di go lang menggunakan [gofpdf](#).

[gofpdf](#) adalah library yang berguna untuk membuat dokumen PDF dari go lang. Penggunaannya tidak terlalu sulit. Jadi mari belajar sambil praktik seperti biasanya.

### C.18.1. Membuat PDF Menggunakan gofpdf

Pertama `go get` library-nya.

```
$ go get -u github.com/jung-kurt/gofpdf
```

Buat folder projek baru, isi main dengan kode berikut.

```
package main

import (
    "github.com/jung-kurt/gofpdf"
    "log"
)

func main() {
    pdf := gofpdf.New("P", "mm", "A4", "")
    pdf.AddPage()
    pdfSetFont("Arial", "B", 16)
    pdf.Text(40, 10, "Hello, world")
    pdf.Image("./sample.png", 56, 40, 100, 0, false, "", 0, "")

    err := pdf.OutputFileAndClose("./file.pdf")
    if err != nil {
        log.Println("ERROR", err.Error())
    }
}
```

Statement `gofpdf.New()` digunakan untuk membuat objek dokumen baru. Fungsi `.New()` tersebut membutuhkan 4 buah parameter.

1. Parameter ke-1, orientasi dokumen, apakah portrait ( `P` ) atau landscape ( `L` ).
2. Parameter ke-2, satuan ukuran yang digunakan, `mm` berarti milimeter.
3. Parameter ke-3, ukuran dokumen, kira pilih A4.
4. Parameter ke-4, path folder font.

Fungsi `.New()` mengembalikan objek PDF. Dari situ kita bisa mengakses banyak method sesuai kebutuhan, beberapa diantaranya adalah 4 buah method yang dicontohkan di atas.

- **Method `.AddPage()`**

Method ini digunakan untuk menambah halaman baru. Defaultnya, objek dokumen yang baru dibuat tidak memiliki halaman. Dengan memanggil `.AddPage()` maka halaman baru dibuat.

Setelah *at least* satu halaman tersedia, kita bisa lanjut ke proses tulis menulis.

- **Method .SetFont()**

Method ini digunakan untuk menge-set konfigurasi font dokumen. Font Family, Font Style, dan Font Size disisipkan dalam parameter secara berurutan.

- **Method .Text()**

Digunakan untuk menulis text pada koordinat tertentu. Pada kode di atas, `40` artinya `40mm` dari kiri, sedangkan `10` artinya `10mm` dari atas. Satuan milimeter digunakan karena pada saat penciptaan objek dipilih `mm` sebagai satuan.

Method ini melakukan penulisan text pada current page.

- **Method .Image()**

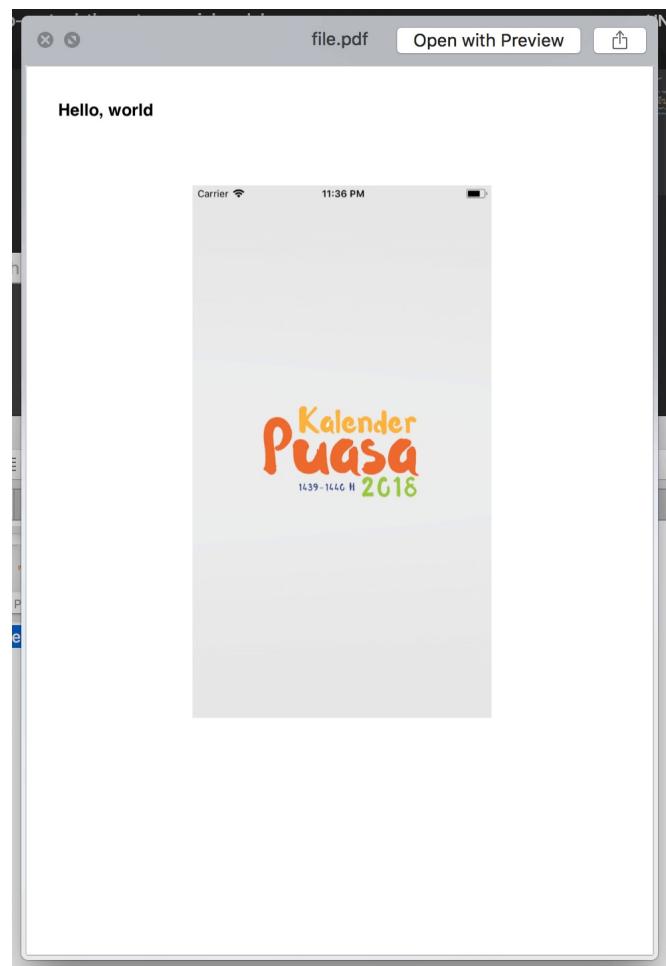
Digunakan untuk menambahkan image. Method ini memerlukan beberapa parameter.

- Parameter ke-1 adalah path image.
- Parameter ke-2 adalah x offset. Nilai `56` artinya `56mm` dari kiri.
- Parameter ke-3 adalah y offset. Nilai `40` artinya `40mm` dari atas.
- Parameter ke-4 adalah width gambar. Jika diisi dengan nilai lebih dari 0 maka gambar akan di-resize secara proporsional sesuai angka. Jika di-isi `0`, maka gambar akan muncul sesuai ukuran aslinya. Pada kode di atas, gambar `sample.png` digunakan, silakan gunakan gambar apa saja bebas.
- Parameter ke-5 adalah height gambar.

Sebenarnya masih banyak lagi method yang tersedia, selengkapnya cek saja di <https://godoc.org/github.com/jung-kurt/gofpdf#Fpdf>.

Setelah selesai bermain dengan objek pdf, gunakan `.outputFileAndClose()` untuk menyimpan hasil sebagai file fisik PDF.

Coba jalankan aplikasi untuk melihat hasilnya. Buka generated file `file.pdf`, isinya kurang lebih seperti gambar berikut.



- 
- [gofpdf](#), by Kurt Jung, MIT license

## C.19. Convert HTML to PDF (go-wkhtmltopdf)

Library gofpdf hanya bisa digunakan untuk pembuatan PDF. Biasanya dalam sebuah aplikasi, report berupa pdf diunduh dengan sumber data adalah halaman web report itu sendiri. Nah, pada bab ini kita akan belajar cara konversi file HTML ke bentuk PDF menggunakan library golang [wkhtmltopdf](#).

Sebenarnya di jaman ini, export HTML to PDF sudah bisa dilakukan di layer front end menggunakan cukup javascript saja. Apalagi jika menggunakan framework terkenal seperti Kendo UI, report yang dimunculkan menggunakan KendoGrid bisa dengan mudah di export ke excel maupun pdf.

Namun pada bab ini akan tetap kita bahas cara tradisional ini, konversi HTML ke PDF pada back end (golang). Semoga berguna.

### C.19.1. Konversi File HTML ke PDF

Buat file html bernama `input.html`, isi dengan apa saja, kalau bisa ada gambarnya juga. Contohnya seperti berikut.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Testing</title>
  </head>
  <body>
    <p>
      Lorem ipsum dolor sit amet, consectetur adipiscing elit.
      Proin tempor ut ipsum et feugiat. Phasellus porttitor,
      felis et gravida aliquam,
      eros orci dignissim magna, at tristique elit massa at magna.
      Etiam et dignissim mi. Phasellus laoreet nulla non aliquam imperdiet.
      Aenean varius turpis at orci posuere, ut bibendum lorem porta.
      Maecenas ullamcorper posuere ante quis ultricies. Aliquam erat volutpat.
      Vestibulum ante ipsum primis in faucibus orci luctus et
      ultrices posuere cubilia Curae;
      Pellentesque eu tellus ante. Vivamus varius nisi non nulla imperdiet,
      vitae pellentesque nibh varius. Fusce diam magna, iaculis eget felis id,
      accumsan convallis elit.
      Phasellus in magna placerat, aliquet ante sed, luctus massa.
      Sed fringilla bibendum feugiat. Suspendisse tempus, purus sit amet
      accumsan consectetur, ipsum odio commodo nisi,
      vel dignissim massa mi ac turpis. Ut fringilla leo ut risus facilisis,
      nec malesuada nunc ornare. Nulla a dictum augue.
    </p>
    
    <!-- other code here -->
  </body>
</html>
```

File html di atas akan kita konversi menjadi sebuah file baru bertipe PDF. Konversi dilakukan menggunakan library wkhtmltopdf. Library ini sebenarnya adalah aplikasi CLI yang dibuat menggunakan bahasa **C++**. Untuk bisa menggunakan kanya kita harus mengunduh lalu meng-install-nya terlebih dahulu.

Silakan unduh installer wkhtmltopdf di <https://wkhtmltopdf.org/downloads.html>, pilih sesuai dengan sistem operasi yang digunakan.

karena wkhtmltopdf merupakan sebuah aplikasi CLI, maka penggunaannya bisa lewat dua cara.

- Cara ke-1: Menggunakan `exec.Command()` untuk mengeksekusi binary. Path file html target disisipkan sebagai argumen command. Silakan merujuk ke referensi [Dasar Pemrograman Golang Bab 46 - Exec](#) untuk mempelajari cara penggunaan exec.
- Cara ke-2: Menggunakan golang wrapper `go-wkhtmltopdf`. Cara ini adalah yang kita pilih.

Secara teknis, `go-wkhtmltopdf` melakukan hal yang sama dengan cara pertama, yaitu mengeksekusi binary `wkhtmltopdf` menggunakan `exec.Command()`.

Mari langsung kita praktikan, buat folder projek baru. Siapkan file main. Isi dengan kode berikut.

```
package main

import (
    "github.com/SebastiaanKlipper/go-wkhtmltopdf"
    "log"
    "os"
)

func main() {
    pdfg, err := wkhtmltopdf.NewPDFGenerator()
    if err != nil {
        log.Fatal(err)
    }

    // ...
}
```

Pembuatan objek PDF dilakukan lewat `wkhtmltopdf.NewPDFGenerator()`. Fungsi tersebut mengembalikan dua buah objek, objek dokumen dan error (jika ada).

Satu objek dokumen merepresentasikan 1 buah file PDF.

Kemudian tambahkan kode untuk membaca file `input.html`. Gunakan `os.open`, agar file tidak langsung dibaca, melainkan dijadikan sebagai `io.Reader`. Masukan objek reader ke fungsi `wkhtmltopdf.NewPageReader()`, lalu sisipkan kembaliannya sebagai page baru di objek PDF.

Kurang lebih kode-nya seperti berikut.

```
f, err := os.Open("./input.html")
if f != nil {
    defer f.Close()
}
if err != nil {
    log.Fatal(err)
}

pdfg.AddPage(wkhtmltopdf.NewPageReader(f))

pdfg.Orientation.Set(wkhtmltopdf.OrientationPortrait)
pdfg.Dpi.Set(300)
```

Method `.AddPage()` milik objek PDF, digunakan untuk menambahkan halaman baru. Halaman baru sendiri dibuat lewat `wkhtmltopdf.NewPageReader()`. Jika ternyata konten pada halaman terlalu panjang untuk dijadikan 1 buah page, maka akan secara otomatis dibuatkan page selanjutnya menyesuaikan konten.

Statement `pdfg.Orientation.Set()` digunakan untuk menentukan orientasi dokumen, apakah portrait atau landscape. Statement `pdfg.Dpi.Set()` digunakan untuk menge-set DPI dokumen.

Gunakan API yang tersedia milik `go-wkhtmltopdf` sesuai kebutuhan. Silakan merujuk ke <https://godoc.org/github.com/SebastiaanKlipper/go-wkhtmltopdf> untuk melihat API apa saja yang tersedia.

Untuk menyimpan objek dokumen menjadi file fisik PDF, ada beberapa step yang harus dilakukan. Pertama, buat dokumen dalam bentuk buffer menggunakan method `.Create()`.

```
err = pdfg.Create()
if err != nil {
    log.Fatal(err)
}
```

Setelah itu, outputkan buffer tersebut sebagai file fisik menggunakan `.WriteFile()`. Sisipkan path destinasi file sebagai parameter method tersebut.

```
err = pdfg.WriteFile("./output.pdf")
if err != nil {
    log.Fatal(err)
}

log.Println("Done")
```

Test aplikasi yang sudah kita buat, lihat hasil generated PDF-nya.



Bisa dilihat, dalam satu PDF dua page muncul, hal ini karena memang isi `input.html` terlalu panjang untuk dijadikan sebagai satu page.

Cara yang kita telah pelajari ini cocok digunakan pada file html yang isinya sudah pasti pada saat file tersebut di-load.

## C.19.2. Konversi HTML dari URL Menjadi PDF

Bagaimana untuk HTML yang sumber nya bukan dari file fisik, melainkan dari URL? tetap bisa dilakukan. Caranya dengan mendaftarkan url sebagai objek page lewat `wkhtmltopdf.NewPage()`, lalu memasukannya ke dalam dokumen sebagai page. Contoh penerapannya bisa dilihat pada kode di bawah ini.

```
pdfg, err := wkhtmltopdf.NewPDFGenerator()
if err != nil {
    log.Fatal(err)
}

pdfg.Orientation.Set(wkhtmltopdf.OrientationLandscape)

page := wkhtmltopdf.NewPage("http://localhost:9000")
page.FooterRight.Set("[page]")
page.FooterFontSize.Set(10)
pdfg.AddPage(page)

err = pdfg.Create()
if err != nil {
    log.Fatal(err)
}

err = pdfg.WriteFile("./output.pdf")
if err != nil {
    log.Fatal(err)
}

log.Println("Done")
```

Cara ini cocok digunakan untuk konversi data HTML yang isinya muncul pada saat page load. Untuk konten-konten yang munculnya asynchronous, seperti di event `document.onload` ada AJAX lalu setelahnya konten baru ditulis, tidak bisa menggunakan cara ini. Solusinya bisa menggunakan teknik export ke PDF dari sisi front end.

- 
- [gopdf](#), by Kurt Jung, MIT license
  - [wkhtmltopdf](#), by Ashish Kulkarni, LGPL-3.0 license
  - [go-wkhtmltopdf](#), by Sebastiaan Klippert, MIT license

## C.20. Scraping & Parsing HTML (goquery)

Golang mempunyai package `net/html`, isinya digunakan untuk keperluan parsing HTML.

Pada bab ini kita akan belajar parsing HTML dengan cara yang lebih mudah, tidak memanfaatkan package `net/html`, melainkan menggunakan [goquery](#). Library ini penggunannya mirip dengan jQuery.

Sebelum dimulai, unduh terlebih dahulu package-nya menggunakan `go get`.

```
$ go get -u github.com/PuerkitoBio/goquery
```

Untuk proses scraping konten html-nya sendiri dilakukan cukup dengan menggunakan fungsi `.Get()` milik package `net/http`.

### C.20.1. Skenario Praktek

Kita akan praktikan penerapan goquery untuk mengambil beberapa data dari website <https://novalagung.com>; pada website tersebut, di halaman landing, ada beberapa blok artikel muncul. Informasi di setiap artikel akan diambil, ditampung dalam satu objek slice, kemudian ditampilkan sebagai JSON string.

**Noval Agung**  
I make humanity better through my inventions



**Connect to Oracle Server using Golang and Go-OCI8 on Ubuntu**

In this post, we'll learn about how to enable connectivity between Oracle Database and Golang on Linux Ubuntu machine. Specific Ubuntu version we are going to use is Ubuntu 12.04.5 LTS,

 5 MIN READ



**Copy Large File From Local to Remote Using Rsync**

In this post, we'll learn how to use rsync to copy large file of data, from local to remote and vice versa. Copy File Using scp Copying file using scp is easy. Below

 2 MIN READ



**Git SVN Clone Since Specific Revision or Latest Revision**

If you have a project that uses SVN and already has so many revisions there, cloning it as git repository could take a lot of time, because all of the revisions will be

 1 MIN READ



**Easy Setup OpenVPN Using Docker DockVPN**

In this post we'll learn how to easily setup OpenVPN using dockvpn. In the simplest terms, VPN creates a secure, encrypted connection, which can be thought of as a tunnel, between your computer

 2 MIN READ



**Setup Ghost v0.11-LTS, Ubuntu, Nginx, Custom Domain, and SSL Let's Encrypt**

Hello guys, In this article we will learn how to setup Ghost blogging platform v0.11-LTS on Ubuntu 16.04 server, with Nginx, as well as set up custom domain and HTTPS; from

 12 MIN READ

Noval Agung © 2018

Latest Posts · Ghost

## C.20.2. Praktek Scraping dan Parsing HTML

Siapkan folder projek baru. Pada file main siapkan sebuah struct dengan nama `Article`, isinya 3 merupakan representasi dari metadata tiap artikel, yaitu `Title`, `URL`, dan `Category`.

```
package main

import (
    "encoding/json"
    "github.com/PuerkitoBio/goquery"
    "log"
    "net/http"
)

type Article struct {
    Title     string
    URL      string
    Category string
}

func main() {
    // code here ...
}
```

Dalam fungsi `main()`, dispatch sebuah client GET request ke url <https://novalagung.com> untuk scraping html-nya.

```
res, err := http.Get("https://novalagung.com")
if err != nil {
    log.Fatal(err)
}
defer res.Body.Close()

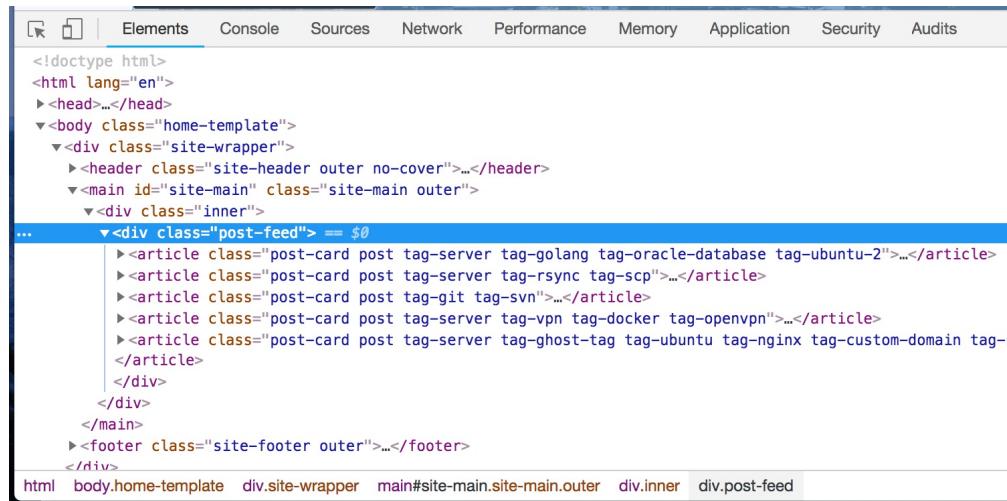
if res.StatusCode != 200 {
    log.Fatalf("status code error: %d %s", res.StatusCode, res.Status)
}
```

Akses property `.Body` dari objek response (yang tipenya adalah reader), masukan sebagai parameter di pemanggilan `goquery.NewDocumentFromReader()`.

```
doc, err := goquery.NewDocumentFromReader(res.Body)
if err != nil {
    log.Fatal(err)
}
```

Statement di atas mengembalikan salah satunya objek `goquery.Document`, yang ditampung dalam `doc`. Kalau dianalogikan dalam jQuery, objek `doc` adalah instance objek hasil `$(selector)`.

Dari objek `goquery.Document`, gunakan API yang tersedia untuk melakukan operasi sesuai kebutuhan. Kita akan ambil semua element artikel sesuai dengan skema html yang ada pada website target.



OK mari langsung kita praktikan, ambil objek `.post-feed` kemudian ambil child elements-nya.

```
rows := make([]Article, 0)

doc.Find(".post-feed").Children().Each(func(i int, sel *goquery.Selection) {
    row := new(Article)
    row.Title = sel.Find(".post-card-title").Text()
    row.URL, _ = sel.Find(".post-card-content-link").Attr("href")
    row.Category = sel.Find(".post-card-tags").Text()
    rows = append(rows, *row)
})

bts, err := json.MarshalIndent(rows, "", " ")
if err != nil {
    log.Fatal(err)
}

log.Println(string(bts))
```

Gunakan `.Find()` untuk mencari elemen, isi parameter dengan selector pencarian. Gunakan `.Each()` untuk me-loop semua elemen yang didapat.

Pada contoh di atas, setelah element `.post-feed` didapatkan, child elements diakses menggunakan `.Children()`.

Method `.Text()` digunakan untuk mengambil text dalam elemen. Sedangkan untuk mengambil value atribut elemen, gunakan `.Attr()`.

Di dalam perulangan, 3 informasi di-ekstrak dari masing-masing elemen artikel, lalu ditampung ke objek `row`, kemudian di-append ke dalam slice.

Di akhir objek slice dikonversi ke bentuk JSON string, lalu ditampilkan.

Jalankan aplikasi, lihat hasilnya.

```
[nvalagung:chapter-b.20 $ go run 1-extract-info.go
2018/08/31 15:44:45 [
{
    "Title": "Connect to Oracle Server using Golang and Go-OCI8 on Ubuntu",
    "URL": "/go-oci8-oracle-linux/",
    "Category": "Server"
},
{
    "Title": "Copy Large File From Local to Remote Using Rsync",
    "URL": "/copy-large-file-using-rsync/",
    "Category": "Server"
},
{
    "Title": "Git SVN Clone Since Specific Revision or Latest Revision",
    "URL": "/git-svn-clone-since-specific-revision-or-latest/",
    "Category": "Git"
},
{
    "Title": "Easy Setup OpenVPN Using Docker DockVPN",
    "URL": "/easy-setup-openvpn-docker/",
    "Category": "Server"
},
{
    "Title": "Setup Ghost v0.11-LTS, Ubuntu, Nginx, Custom Domain, and SSL Let's Encrypt",
    "URL": "/ghost-v011-lts-ubuntu-nginx-custom-domain-ssl/",
    "Category": "Server"
}
]
nvalagung:chapter-b.20 $ ]
```

### C.20.3. Modifikasi HTML

API yang tersedia dalam goquery tidak hanya untuk keperluan ekstraksi informasi, tapi bisa juga untuk modifikasi/manipulasi HTML, dan operasi lainnya.

Mari kita langsung praktekan saja. Buat file main baru, siapkan string html. Kali ini sumber data bukan berasal dari website asli, melainkan dari string html.

```
package main

import (
    "github.com/PuerkitoBio/goquery"
    "github.com/yosssi/gohtml"
    "log"
    "strings"
)

const sampleHTML = `<!DOCTYPE html>
<html>
    <head>
        <title>Sample HTML</title>
    </head>
    <body>
        <h1>Header</h1>
        <footer class="footer-1">Footer One</footer>
        <footer class="footer-2">Footer Two</footer>
        <footer class="footer-3">Footer Three</footer>
    </body>
</html>
```

HTML string di atas dijadikan ke bentuk `goquery.Document` lewat fungsi `goquery.NewDocumentFromReader()`. Karena parameter yang dibutuhkan bertipe reader, maka konversi string html ke tipe reader lewat `strings.NewReader()`.

```
doc, err := goquery.NewDocumentFromReader(strings.NewReader(sampleHTML))
if err != nil {
```

```
    log.Fatal(err)
}
```

Selanjutnya, lakukan beberapa modifikasi.

```
doc.Find("h1").AfterHtml("<p>Lorem Ipsum Dolor Sit Amet Gedhang Goreng</p>")
doc.Find("p").AppendHtml(" <b>Tournesol</b>")
doc.Find("h1").SetAttr("class", "header")
doc.Find("footer").First().Remove()
doc.Find("body > *:nth-child(4)").Remove()
```

Berikut penjelasan beberapa API yang digunakan pada kode di atas.

- Method `.AfterHtml()`, digunakan untuk menambahkan elemen baru, posisinya ditempatkan setelah elemen objek pemanggilan method.
- Method `.AppendHtml()`, digunakan untuk menambahkan child elemen baru.
- Method `.SetAttr()`, digunakan untuk menambahkan/mengubah atribut elemen.
- Method `.First()`, digunakan untuk mengakses elemen pertama. Pada kode di atas `doc.Find("footer")` mengembalikan semua footer yang ada, sesuai selector. Pengaksesan method `.First()` menjadikan hanya elemen footer pertama yang diambil.
- Method `.Remove()`, digunakan untuk menghapus current element.

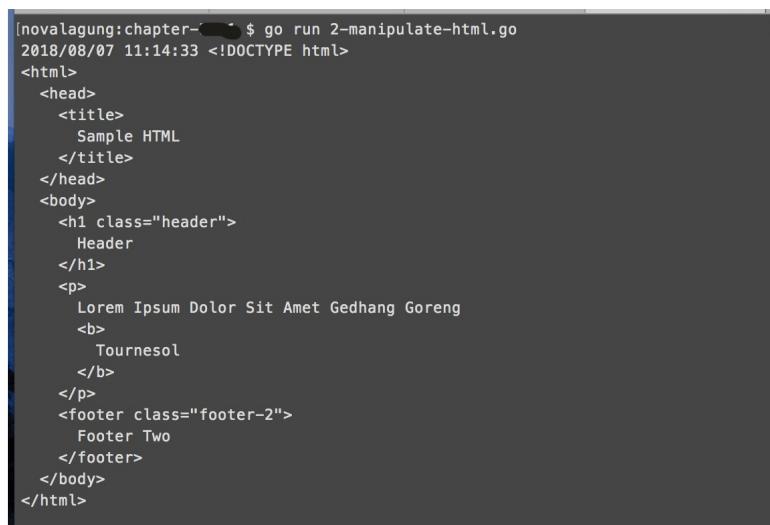
Ambil bentuk string html dari objek `doc` yang sudah banyak dimodifikasi. Jangan gunakan `doc.Html()` karena yang dikembalikan adalah [inner html](#). Gunakan `goqueryOuterHtml(doc.Selection)` agar yang dikembalikan [outer html](#)-nya.

```
modifiedHTML, err := goqueryOuterHtml(doc.Selection)
if err != nil {
    log.Fatal(err)
}

log.Println(gohtml.Format(modifiedHTML))
```

Pada kode di atas kita menggunakan satu lagi library, [gohtml](#). Fungsi `.Format()` dalam library tersebut digunakan untuk code beautification.

Jalankan aplikasi, lihat hasilnya.



```
[inavalagung:chapter-] $ go run 2-manipulate-html.go
2018/08/07 11:14:33 <!DOCTYPE html>
<html>
  <head>
    <title>
      Sample HTML
    </title>
  </head>
  <body>
    <h1 class="header">
      Header
    </h1>
    <p>
      Lorem Ipsum Dolor Sit Amet Gedhang Goreng
      <b>
        Tournesol
      </b>
    </p>
    <footer class="footer-2">
      Footer Two
    </footer>
  </body>
</html>
```

- 
- [goquery](#), by Martin Angers, BSD-3-Clause license

- [gohtml](#), by Keiji Yoshida, MIT license

## C.21. Parse & Generate XML (etree)

Pada bab ini kita akan belajar cara parsing file xml, dan cara membuat xml baru. Library yang digunakan adalah [etree](#), silakan `go get` terlebih dahulu.

```
$ go get -u github.com/beevik/etree
```

### C.21.1. Membaca dan Parsing File XML

Mari langsung kita praktikan, siapkan folder projek baru. Buat satu buah file `data.xml`, isinya sebagai berikut.

```
<?xml version="1.0" encoding="UTF-8"?>
<website>
    <title>Noval Agung</title>
    <url>https://novalagung.com</url>
    <contents>
        <article>
            <category>Server</category>
            <title>Connect to Oracle Server using Golang and Go-OCI8 on Ubuntu</title>
            <url>/go-oci8-oracle-linux/</url>
        </article>
        <article>
            <category>Server</category>
            <title>Easy Setup OpenVPN Using Docker DockVPN</title>
            <url>/easy-setup-openvpn-docker/</url>
        </article>
        <article info="popular article">
            <category>Server</category>
            <title>Setup Ghost v0.11-LTS, Ubuntu, Nginx, Custom Domain, and SSL</title>
            <url>/ghost-v011-lts-ubuntu-nginx-custom-domain-ssl/</url>
        </article>
    </contents>
</website>
```

Silakan perhatikan xml di atas, akan kita ambil semua element `article` beserta isinya, untuk kemudian ditampung dalam slice.

Buat file main, didalamnya, buat objek dokumen bertipe `etree.Document` lewat fungsi `etree.NewDocument()`. Dari objek tersebut, baca file xml yang sudah dibuat, gunakan method `.ReadFromFile()` untuk melakukan proses baca file.

```
package main

import (
    "encoding/json"
    "github.com/beevik/etree"
    "log"
)

type M map[string]interface{}

func main() {
    doc := etree.NewDocument()
    if err := doc.ReadFromFile("./data.xml"); err != nil {
        log.Fatal(err.Error())
    }

    // ...
}
```

Dari objek `doc`, ambil root element `<website/>`, lalu akses semua element `<article/>` kemudian lakukan perulangan. Di dalam tiap perulangan, ambil informasi `title`, `url`, dan `category`, tumpung sebagai element slice `rows`.

```
root := doc.SelectElement("website")
rows := make([]M, 0)

for _, article := range root.FindElements("//article") {
    row := make(M)
    row["title"] = article.SelectElement("title").Text()
    row["url"] = article.SelectElement("url").Text()

    categories := make([]string, 0)
    for _, category := range article.SelectElements("category") {
        categories = append(categories, category.Text())
    }
    row["categories"] = categories

    if info := article.SelectAttr("info"); info != nil {
        row["info"] = info.Value
    }

    rows = append(rows, row)
}
```

Objek element menyediakan beberapa method untuk keperluan seleksi dan pencarian element. Empat diantaranya sebagai berikut.

- Method `.SelectElement()`, untuk mengambil satu buah child element sesuai selector.
- Method `.SelectElements()`, sama seperti `.SelectElement()`, perbedannya yang dikembalikan adalah semua child elements (sesuai selector).
- Method `.FindElement()`, untuk mencari elements dalam current element, bisa berupa child, grand child, atau level yang lebih dalam, sesuai selector. Yang dikembalikan satu buah element saja.
- Method `.FindElements()`, sama seperti `.FindElement()`, perbedannya yang dikembalikan adalah semua elements (sesuai selector).

Pada kode di atas, hasil dari statement `root.FindElements("//article")` di looping. Statement tersebut mengembalikan banyak element sesuai selector pencarian. Arti selector `//article` sendiri adalah melakukan pencarian element dengan nama `article` secara rekursif.

Di tiap perulangan, child element `title` dan `url` diambil. Gunakan method `.Text()` untuk mengambil isi element.

Sedangkan pada element `<category/>` pencarian child elements dilakukan menggunakan method `.SelectElements()`, karena beberapa artikel memiliki lebih dari satu category.

Untuk mengakses value dari atribut, gunakan method `.SelectAttr()`.

Setelah perulangan selesai, data artikel ada dalam objek `rows`. Tampilkan isinya sebagai JSON string.

```
bts, err := json.MarshalIndent(rows, "", " ")
if err != nil {
    log.Fatal(err)
}

log.Println(string(bts))
```

Jalankan aplikasi, lihat hasilnya.

```
[novalagung:chapter-] $ go run 1-find-all.go
2018/08/08 12:17:16 [
{
  "categories": [
    "Server"
  ],
  "title": "Connect to Oracle Server using Golang and Go-OCI8 on Ubuntu",
  "url": "/go-oci8-oracle-linux/"
},
{
  "categories": [
    "Server"
  ],
  "title": "Easy Setup OpenVPN Using Docker DockVPN",
  "url": "/easy-setup-openvpn-docker/"
},
{
  "categories": [
    "Server"
  ],
  "info": "popular article",
  "title": "Setup Ghost v0.11-LTS, Ubuntu, Nginx, Custom Domain, and SSL",
  "url": "/ghost-v011-lts-ubuntu-nginx-custom-domain-ssl/"
}
]
```

## C.21.2. XML Query

XQuery atau XML Query adalah bahasa query untuk pengolahan XML. Spesifikasinya bisa dilihat di <https://www.w3.org/TR/xquery-31>.

Pada pembahasan di atas kita menggunakan query `//article` untuk melakukan pencarian semua element artikel secara rekursif.

Berikut adalah contoh lain implementasi xquery yang lebih kompleks.

```
popularArticleText := root.FindElement(`//article[@info='popular article']/title`)
if popularArticleText != nil {
    log.Println("Popular article", popularArticleText.Text())
}
```

Penjelasan mengenai xquery `//article[@info='popular article']/title` dipecah menjadi 3 tahap agar mudah untuk dipahami.

1. Selector bagian `//article`, artinya dilakukan pencarian rekursif dengan kriteria: element bernama `article`.
2. Selector bagian `[@info='popular article']`, artinya dilakukan pencarian dengan kriteria: element memiliki atribut `info` yang berisi `popular article`.
3. Selector bagian `/title`, artinya dilakukan pencarian child element dengan kriteria: element bernama `title`.

Jika 3 penjelasan bagian di atas digabungkan, maka kurang lebih arti dari `//article[@info='popular article']/title` adalah, dilakukan pencarian secara rekursif dengan kriteria adalah: element bernama `article` dan harus memiliki atribut `info` yang berisi `popular article`, setelah diketemukan, dicari child element-nya menggunakan kriteria: element bernama `title`.

Berikut adalah hasil dari query di atas.

```
[novalagung:chapter-] $ go run 2-advanced-find.go
2018/08/08 12:52:21 Popular article Setup Ghost v0.11-LTS, Ubuntu, Nginx, Custom D
```

Silakan coba explore sendiri mengenai xquery untuk contoh lainnya.

## C.21.3. Membuat XML dari Golang

Di atas kita telah mempelajari cara baca XML; kali ini kita akan coba buat file XML menggunakan etree. Informasi yang akan ditulis ke file xml datanya bersumber dari JSON string (yang nantinya di-decode terlebih dahulu ke bentuk objek sebelum digunakan).

Siapkan file baru, buat struct `Document`. Nantinya sebuah objek dicetak lewat struk ini, tugasnya sendiri adalah menampung data hasil proses decoding json.

```
package main

import (
    "encoding/json"
    "github.com/beevik/etree"
    "log"
)

type Document struct {
    Title    string
    URL     string
    Content struct {
        Articles []struct {
            Title    string
            URL     string
            Categories []string
            Info     string
        }
    }
}

func main () {
    // code here
}
```

Siapkan JSON string.

```
const jsonString = `{
    "Title": "Noval Agung",
    "URL": "https://novalagung.com",
    "Content": {
        "Articles": [
            {
                "Categories": [ "Server" ],
                "Title": "Connect to Oracle Server using Golang and Go-OCI8 on Ubuntu",
                "URL": "/go-oci8-oracle-linux/"
            },
            {
                "Categories": [ "Server", "VPN" ],
                "Title": "Easy Setup OpenVPN Using Docker DockVPN",
                "URL": "/easy-setup-openvpn-docker/"
            },
            {
                "Categories": [ "Server" ],
                "Info": "popular article",
                "Title": "Setup Ghost v0.11-LTS, Ubuntu, Nginx, Custom Domain, and SSL",
                "URL": "/ghost-v011-lts-ubuntu-nginx-custom-domain-ssl/"
            }
        ]
    }
}`
```

Decode JSON string di atas ke objek cetakan `Document`.

```
data := Document{}
err := json.Unmarshal([]byte(jsonString), &data)
if err != nil {
    log.Fatal(err.Error())
}
```

Selanjutnya buat objek etree baru, siapkan root element `website`. Di dalamnya buat 2 child elements: `title` dan `url`, nilai masing-masing didapat dari objek `data`.

```
doc := etree.NewDocument()
doc.CreateProcInst("xml", `version="1.0" encoding="UTF-8"`)
website := doc.CreateElement("website")
website.CreateElement("title").SetText(data.Title)
website.CreateElement("url").SetText(data.URL)
```

Method `.CreateElement()` digunakan untuk membuat child element baru. Pemanggilannya disertai dengan satu parameter, yang merupakan representasi dari nama element yang ingin dibuat.

Method `.SetText()` digunakan untuk menge-set nilai element.

Siapkan satu element lagi dibawah root, namanya `contents`. Loop objek slice artikel, dan di tiap perulangannya, buat element dengan nama `article`, sisipkan sebagai child `contents`.

```
content := website.CreateElement("contents")

for _, each := range data.Content.Articles {
    article := content.CreateElement("article")
    article.CreateElement("title").SetText(each.Title)
    article.CreateElement("url").SetText(each.URL)

    for _, category := range each.Categories {
        article.CreateElement("category").SetText(category)
    }

    if each.Info != "" {
        article.CreateAttr("info", each.Info)
    }
}
```

Khusus untuk objek artikel yang property `.Info`-nya tidak kosong, buat atribut dengan nama `info` pada element `article` yang bersangkutan, simpan nilai property sebagai nilai atribut tersebut.

Terakhir simpan objek dokumen etree sebagai file.

```
doc.Indent(2)

err = doc.WriteToFile("output.xml")
if err != nil {
    log.Println(err.Error())
}
```

Method `.Indent()` di atas digunakan untuk menentukan indentasi element dalam file.

Jalankan aplikasi, lihat hasilnya.

```
[novalagung:chapter-] $ go run 3-create-xml.go
[novalagung:chapter-] $ cat output.xml
<?xml version="1.0" encoding="UTF-8"?>
<website>
  <title>Noval Agung</title>
  <url>https://novalagung.com</url>
  <contents>
    <article>
      <title>Connect to Oracle Server using Golang and Go-OCI8 on Ubuntu</title>
      <url>/go-oci8-oracle-linux/</url>
      <category>Server</category>
    </article>
    <article>
      <title>Easy Setup OpenVPN Using Docker DockVPN</title>
      <url>/easy-setup-openvpn-docker/</url>
      <category>Server</category>
      <category>VPN</category>
    </article>
    <article info="popular article">
      <title>Setup Ghost v0.11-LTS, Ubuntu, Nginx, Custom Domain, and SSL</title>
      <url>/ghost-v011-lts-ubuntu-nginx-custom-domain-ssl/</url>
      <category>Server</category>
    </article>
  </contents>
</website>
```

- 
- [etree](#), by Brett Vickers, BSD-2-Clause license

## C.22. HTTPS/TLS Web Server

Pada bagian ini kita akan belajar cara meng-enable fasilitas SSL/TLS pada web server.

### C.22.1. Definisi

- **SSL**

**SSL, Secure Sockets Layer**, adalah standar untuk pengamanan komunikasi lewat internet. Data atau informasi yang sedang dikomunikasikan dari sebuah system ke system lain akan di-proteksi, dengan cara adalah mengacak informasi tersebut menggunakan algoritma enkripsi.

- **SSL Certificates**

**SSL Certificate**, adalah sebuah file berisikan informasi mengenai website, yang nantinya dibutuhkan untuk enkripsi data. SSL Certificate berisi **Public Key**. Public key digunakan untuk meng-enkripsi data yang akan di transfer.

Certificate ditandatangi secara digital oleh **Certificate Authorities (CA)**. Digital Signature atau tanda tangan digital merupakan sebuah kode unik yang di-generate dengan teknologi cryptography (**Public Key Infrastructure**).

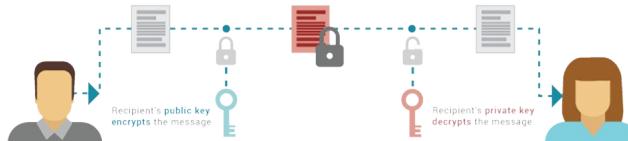
Certificate Authorities sendiri merupakan entitas atau institusi legal yang mengeluarkan dan mem-verifikasi sertifikat digital.

Ketika seorang pengguna internet surfing, mengakses sebuah website yang website tersebut menerapkan SSL, informasi yang dikirim akan di encrypt dengan aman (menggunakan public key) dan hanya bisa di-decrypt menggunakan **Private Key**.

Private Key, atau Secret Key, adalah file terpisah yang diperlukan pada proses dekripsi data yang di-encrypt menggunakan public key.

Berikut merupakan penjelasan dalam bentuk gambar yang diambil dari [coinjolt.com](http://coinjolt.com).

### PUBLIC KEY CRYPTOGRAPHY



Kedua file certificate dan file private key harus disimpan dengan sangat super aman di server.

- **TLS**

**TLS, Transport Layer Security**, adalah versi yang lebih update dari SSL.

- **HTTPS**

**HTTPS, Hyper Text Transfer Protocol Secure**, adalah ekstensi dari HTTP yang berguna untuk pengamanan komunikasi lewat internet. Data atau informasi yang dikomunikasikan di-enkripsi menggunakan **TLS**.

## C.22.2. Generate Private Key & Public Key Menggunakan openssl

Untuk menerapkan TLS pada web server aplikasi golang, private key dan public key perlu kita siapkan terlebih dahulu.

Gunakan `openssl` untuk generate private key.

```
$ openssl genrsa -out server.key 2048
$ openssl ecparam -genkey -name secp384r1 -out server.key
```

Dua command di atas menghasilkan `server.key` yang merupakan private key. Setelah itu, generate *self-signed* certificate (yang berisikan public key) dari private key yang telah dibuat.

```
$ openssl req -new -x509 -sha256 -key server.key -out server.crt -days 3650
```

Command untuk generate certificate di atas akan memunculkan form. Informasi seperti alamat, email, host diminta untuk di-isi, pastikan isi dengan benar, terutama di bagian "**Common Name**" dan **Email Address**.

- Pada bagian common name isi dengan **localhost**.
- Untuk email isi dengan alamat email yang valid.

Tampilannya kurang lebih seperti pada screenshot berikut.

```
[novalagung:certs $ openssl genrsa -out server.key 2048
Generating RSA private key, 2048 bit long modulus
.....+
.....+
e is 65537 (0x10001)
[novalagung:certs $ openssl ecparam -genkey -name secp384r1 -out server.key
[novalagung:certs $ openssl req -new -x509 -sha256 -key server.key -out server.crt -days 3650
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) []:ID
State or Province Name (full name) []:Jawa Timur
Locality Name (eg, city) []:Surabaya
Organization Name (eg, company) []:Noval Agung
Organizational Unit Name (eg, section) []:Noval Agung
Common Name (eg, fully qualified host name) []:localhost
Email Address []:caknopal@gmail.com
[novalagung:certs $ ls
server.crt    server.key
novalagung:certs $ ]
```

Selain `.crt` dan `.key`, ada ekstensi lain lagi seperti `.pem`. Format `.pem` ini merupakan jenis encoding yang sangat sering digunakan pada file kriptografi sejenis `.key` dan `.crt`. File `.crt` dan `.key` bisa di konversi ke `.pem`, dan juga sebaliknya.

## C.22.3. Project Structure

Buat sebuah projek folder, copy 2 file yang telah ter-generate ke dalamnya. Lalu siapkan file `main.go`.



## C.22.4. Web Servers

Pada `main.go`, siapkan sebuah fungsi `StartNonTLSServer()`, berisikan mux dengan satu buah routing, untuk redirect request dari protokol `http` ke `https`. Nantinya semua request yang mengarah ke `http://localhost` di-redirect ke `https://localhost`. Start mux ini pada port `:80`.

```
package main

import (
    "log"
    "net/http"
)

func StartNonTLSServer() {
    mux := new(http.ServeMux)
    mux.Handle("/", http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
        log.Println("Redirecting to https://localhost/")
        http.Redirect(w, r, "https://localhost/", http.StatusTemporaryRedirect)
    }))
    http.ListenAndServe(":80", mux)
}
```

Lalu pada fungsi `main()`, buat mux baru lagi, dengan isi satu buah routing, menampilkan text "Hello World!". Start mux menggunakan `http.ListenAndServeTLS()` pada port `:443`, tak lupa sisipkan path dari file private key dan public key sebagai argumen fungsi.

```
func main() {
    go StartNonTLSServer()

    mux := new(http.ServeMux)
    mux.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
        w.Write([]byte("Hello World!"))
    })

    log.Println("Server started at :443")
    err := http.ListenAndServeTLS(":443", "server.crt", "server.key", mux)
    if err != nil {
        panic(err)
    }
}
```

Tak lupa fungsi `StartNonTLSServer()` juga dipanggil dalam `main()`. Jadi pada satu program terdapat 2 buah web server dijalankan.

OK, jalankan aplikasi.

Jika error `panic: listen tcp :443: bind: permission denied` muncul, coba jalankan aplikasi menggunakan `sudo`, contoh: `sudo go run main.go`

## C.22.5. Testing

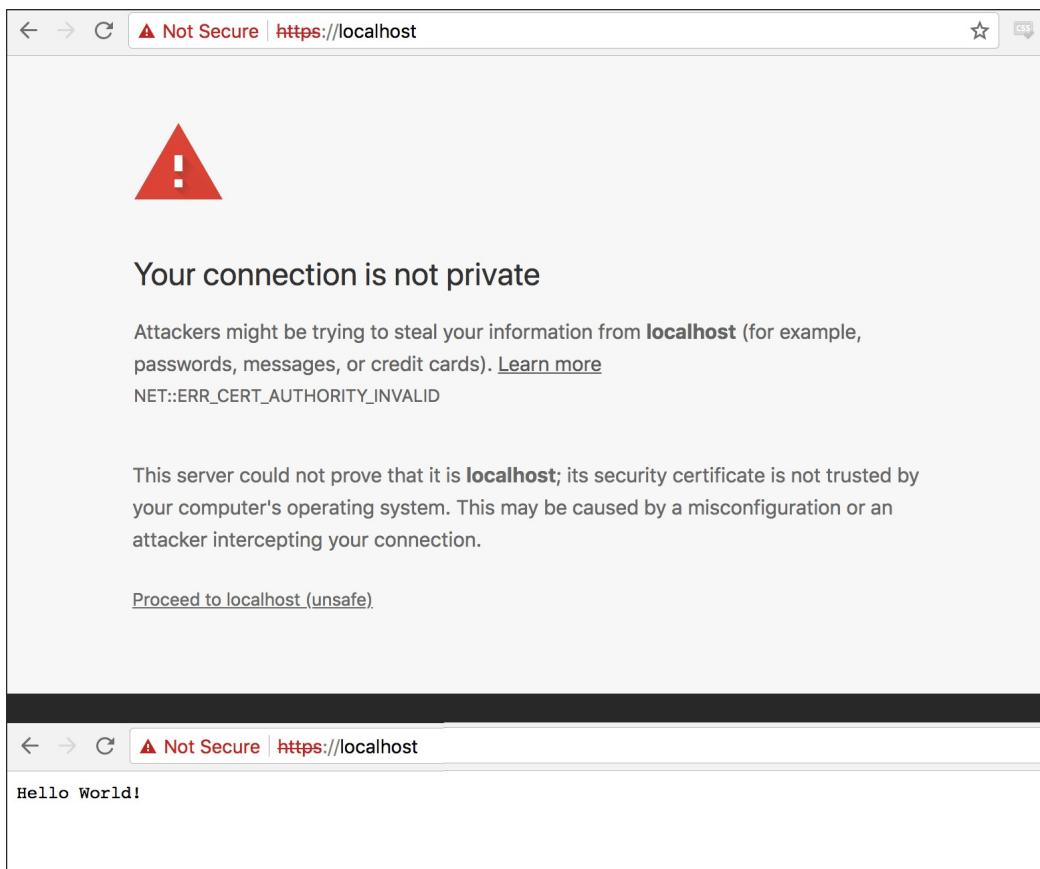
Test aplikasi menggunakan `curl`. Untuk request ke protokol `https` coba tambahkan flag `--insecure` untuk men-disable verifikasi certificate.

```
[novalagung:chapter-] $ curl -I http://localhost
HTTP/1.1 307 Temporary Redirect
Content-Type: text/html; charset=utf-8
Location: https://localhost/
Date: Fri, 08 Jun 2018 10:05:31 GMT

[novalagung:chapter-] $ curl --insecure https://localhost
Hello World!
```

Coba test juga menggunakan browser, jika menggunakan chrome maka akan muncul warning

`NET::ERR_CERT_AUTHORITY_INVALID` Klik **Advanced → Proceed to localhost (unsafe)**.



Warning `NET::ERR_CERT_AUTHORITY_INVALID` muncul ketika mengakses sebuah website menggunakan protokol `https` yang dimana website ini mengaplikasikan **self-signed certificate**, bukan menggunakan certificate yang sudah diverifikasi oleh CA.

## C.23. HTTP/2 dan HTTP/2 Server Push

HTTP/2 adalah versi terbaru protokol HTTP, dikembangkan dari protokol [SPDY](#) yang diinisiasi oleh Google.

Protokol ini sekarang sudah kompatibel dengan banyak browser diantaranya: Chrome, Opera, Firefox 9, IE 11, Safari, Silk, dan Edge.

Kelebihan HTTP/2 dibanding HTTP 1.1 (protokol yang umumnya digunakan) sebagian besar adalah pada performa dan sekuriti. Berikut merupakan beberapa point yang menjadi kelebihan dari protokol baru ini.

- Backward compatible dengan HTTP 1.1
- Kompresi data pada HTTP Headers
- Multiplexing banyak request (dalam satu koneksi TCP)
- HTTP/2 Server Push

Pada bab ini kita akan belajar cara menerapkan HTTP/2 dan salah satu fitur milik protokol ini yaitu HTTP/2 Server Push.

Mengenai multiplexing banyak request tidak akan kita bahas pada buku ini, silakan coba pelajari sendiri jika tertarik, menggunakan library cmux.

### C.23.1. HTTP/2 di Golang

Golang memiliki dukungan sangat baik terhadap HTTP/2. Dengan cukup meng-enable fasilitas TLS/HTTPS maka aplikasi golang secara otomatis menggunakan HTTP/2.

Untuk memastikan mari kita langsung praktikkan, coba duplikat projek pada bab sebelumnya ([A.23. HTTPS/TLS Web Server](#)) sebagai projek baru, jalankan aplikasinya lalu cek di browser chrome. Gunakan chrome extension [HTTP/2 and SPDY indicator](#) untuk menge-test apakah HTTP/2 sudah enabled.



Perlu diketahui untuk golang versi sebelum **1.6** ke bawah, secara default HTTP/2 tidak akan di-enable. Perlu memanggil fungsi `http2.ConfigureServer()` secara eksplisit untuk meng-enable HTTP/2. Fungsi tersebut tersedia dalam package `golang.org/x/net/http2`. Lebih jelasnya silakan baca [laman dokumentasi](#).

### C.23.2. HTTP/2 Server Push

HTTP/2 Server Push adalah salah satu fitur pada HTTP/2, berguna untuk mempercepat response dari request, dengan cara data yang akan di-response dikirim terlebih dahulu oleh server.

Fitur server push ini cocok digunakan untuk push data assets, seperti: css, gambar, js, dan file assets lainnya.

Lalu apakah server push ini bisa dimanfaatkan untuk push data JSON, XML, atau sejenisnya? Sebenarnya bisa, hanya saja ini akan menyalahi tujuan dari penciptaan server push sendiri dan hasilnya tidak akan optimal, karena sebenarnya server push tidak murni bidirectional, masih perlu adanya request ke server untuk mendapatkan data yg sudah di push oleh server itu sendiri.

HTTP/2 server push bukanlah pengganti dari websocket. Gunakan websocket untuk melakukan komunikasi bidirectional antara server dan client.

Untuk mengecek support-tidaknya server push, lakukan casting pada objek `http.ResponseWriter` milik handler ke interface `http.Pusher`, lalu manfaatkan method `Push()` milik interface ini untuk push data dari server.

Fasilitas server push ini hanya bisa digunakan pada golang versi 1.8 ke atas.

## C.23.3. Praktek

Mari kita praktikan. Buat projek baru, buat file `main.go`, isi dengan kode berikut.

```
package main

import (
    "fmt"
    "log"
    "net/http"
)

func main() {
    http.Handle("/static/",
        http.StripPrefix("/static/",
            http.FileServer(http.Dir(".")))

    log.Println("Server started at :9000")
    err := http.ListenAndServeTLS(":9000", "server.crt", "server.key", nil)
    if err != nil {
        panic(err)
    }
}
```

Dalam folder proyek baru di atas, siapkan juga beberapa file lain:

- File `app.js`
- File `app.css`
- File `server.crt`, salin dari proyek sebelumnya
- File `server.key`, salin dari proyek sebelumnya

Selanjutnya siapkan string html, nantinya akan dijadikan sebagai output route `/`.

```
const indexHTML = `

<!DOCTYPE html>
<html>
    <head>
        <title>Hello World</title>
        <script src="/static/app.js"></script>
        <link rel="stylesheet" href="/static/app.css">
    </head>
    <body>
        Hello, gopher!<br>
        
    </body>
</html>
`
```

Siapkan route `/`, render string html tadi sebagai output dari endpoint ini.

```
http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
    if r.URL.Path != "/" {
        http.NotFound(w, r)
```

```

        return
    }

    if pusher, ok := w.(http.Pusher); ok {
        if err := pusher.Push("/static/app.js", nil); err != nil {
            log.Printf("Failed to push: %v", err)
        }

        if err := pusher.Push("/static/app.css", nil); err != nil {
            log.Printf("Failed to push: %v", err)
        }
    }

    fmt.Fprintf(w, indexHTML)
}

```

Pada handler di atas bisa kita lihat bahwa selain me-render string html, rute ini juga memiliki tugas lain yaitu push assets.

Cara untuk mendeteksi apakah server push di-support, adalah dengan meng-casting objek `http.ResponseWriter` ke tipe `http.Pusher`. Proses casting tersebut mengembalikan dua buah data.

1. Data objek yang sudah di casting.
2. Variabel bertipe `bool` penanda aplikasi kita mendukung mendukung server push atau tidak.

Jika nilai variabel `ok` adalah `true`, maka server push di-support.

Method `Push()` milik `http.Pusher` digunakan untuk untuk push data. Endpoint yang disisipkan sebagai argumen, datanya akan di push ke front end oleh server, dalam contoh di atas adalah `/static/app.js` dan `/static/app.css`.

Server push ini adalah per endpoint atau rute. Jika ada rute lain, maka dua assets di atas tidak akan di push, kecuali method `Push()` dipanggil lagi dalam rute lain tersebut.

Daripada memanggil method push satu-per-satu pada banyak handler, lebih baik jadikan sebagai middleware terpisah.

Kegunaan dari fungsi `fmt.Fprintf()` adalah untuk render html, sama seperti `w.Write()`.

OK, jalankan aplikasi lalu test.

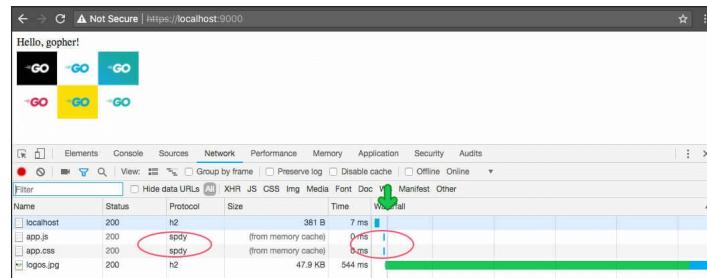
## C.23.4. Testing

Perbedaan antara aplikasi yang menerapkan HTTP/2 dan tidak, atau yang menerapkan server push atau tidak; adalah tidak terasa bedanya jika hanya di-test lewat lokal saja.

Untuk mengecek HTTP/2 diterapkan atau tidak, kita bisa gunakan Chrome extension **HTTP/2 and SPDY indicator**.

Untuk mengecek server push pada tiap request sebenarnya bisa hanya cukup menggunakan chrome dev tools, namun fitur ini hanya tersedia pada [Chrome Canary](#). Download browser tersebut lalu install, gunakan untuk testing aplikasi kita.

Pada saat mengakses `https://localhost:9000` pastikan developer tools sudah aktif (klik kanan, inspect element), lalu buka tab **Network**.



Untuk endpoint yang menggunakan server push, pada kolom **Protocol** nilainya adalah **spdy**. Pada screenshot di atas terlihat bahwa assets `app.js` dan `app.css` dikirim lewat server push.

Jika kolom Protocol tidak muncul, klik kanan pada kolom, lalu centang **Protocol**.

Berikut merupakan variasi nilai pada kolom protocol.

- **http/1.1**, protokol yang kita gunakan mulai tahun 1999.
- **spdy**, versi pertama spesifikasi HTTP/2 dari google, mengawali terciptanya HTTP/2.
- **h2**, kependekan dari "HTTP 2".
- **h2c**, kependekan dari "HTTP 2 Cleartext". HTTP/2 lewat kanal yang tidak ter-enkripsi.

Selain dari kolom protocol, penanda server push bisa dilihat juga lewat grafik **Waterfall**. Garis warna ungu muda pada grafik tersebut adalah start time. Untuk endpoint yang di push maka bar chart akan muncul sebelum garis ungu atau start time.

## C.24. Client HTTP Request

Pada bab ini kita akan belajar tentang topik yang sedikit berbeda dibanding bab sebelumnya, yaitu cara untuk melakukan http request ke sebuah web server.

Dua aplikasi akan dibuat, server dan client. Server merupakan aplikasi web server kecil, memiliki satu endpoint. Lalu dari client http request di-trigger, dengan tujuan adalah server.

### C.24.1. Aplikasi Server

Buat projek baru seperti biasa, lalu buat `server.go`. Import package yang dibutuhkan.

```
package main

import (
    "encoding/json"
    "fmt"
    "log"
    "net/http"
)

type M map[string]interface{}
```

Pada fungsi `main()`, siapkan mux dengan isi satu buah handler, jalankan pada port `:9000`.

```
func main() {
    mux := new(http.ServeMux)
    mux.HandleFunc("/data", ActionData)

    server := new(http.Server)
    server.Handler = mux
    server.Addr = ":9000"

    log.Println("Starting server at", server.Addr)
    err := server.ListenAndServe()
    if err != nil {
        log.Fatalln("Failed to start web server", err)
    }
}
```

Buat fungsi `ActionData()` yang merupakan handler dari rute `/data`. Handler ini hanya menerima method `POST`, dan mewajibkan consumer endpoint untuk menyisipkan payload dalam request-nya, dengan isi adalah JSON.

```
func ActionData(w http.ResponseWriter, r *http.Request) {
    log.Println("Incoming request with method", r.Method)

    if r.Method != "POST" {
        http.Error(w, "Method not allowed", http.StatusBadRequest)
        return
    }

    payload := make(M)
    err := json.NewDecoder(r.Body).Decode(&payload)
    if err != nil {
        http.Error(w, err.Error(), http.StatusBadRequest)
        return
    }
}
```

```

if _, ok := payload["Name"]; !ok {
    http.Error(w, "Payload `Name` is required", http.StatusBadRequest)
    return
}

// ...
}

```

Isi dari `r.Body` kita decode ke objek `payload` yang bertipe `map[string]interface{}`. Setelah proses decoding selesai, terdapat pengecekan ada tidaknya property `Name` dalam payload. Jika tidak ada maka dianggap bad request.

Setelah itu, buat objek `data` dengan 2 property, yang salah satunya berisi kombinasi string dari payload `.Name`.

```

data := M{
    "Message": fmt.Sprintf("Hello %s", payload["Name"]),
    "Status":  true,
}

w.Header().Set("Content-Type", "application/json")
err = json.NewEncoder(w).Encode(data)
if err != nil {
    http.Error(w, err.Error(), http.StatusInternalServerError)
}

```

Cara render output JSON biasanya kita lakukan menggunakan statement `w.Write()` dengan isi adalah `[]byte` milik JSON. Ada cara lain, yaitu menggunakan json encoder. Penerapannya bisa dilihat pada kode di atas.

Aplikasi server sudah siap. Selanjutnya kita masuk ke bagian pembuatan aplikasi client.

## C.24.2. Aplikasi Client

Tugas dari aplikasi client: melakukan http request ke aplikasi server, pada endpoint `/data` sesuai dengan spesifikasi yang sudah dijelaskan di atas (ber-method POST, dan memiliki JSON payload).

Buat file baru, `client.go`, import package yang diperlukan.

```

package main

import (
    "bytes"
    "encoding/json"
    "log"
    "net/http"
)

type M map[string]interface{}

```

Buat fungsi `doRequest()`. Fungsi ini kita gunakan men-trigger http request.

```

func doRequest(url, method string, data interface{}) (interface{}, error) {
    var payload *bytes.Buffer = nil

    if data != nil {
        payload = new(bytes.Buffer)
        err := json.NewEncoder(payload).Encode(data)
        if err != nil {
            return nil, err
        }
    }
}

```

```

request, err := http.NewRequest(method, url, payload)
if err != nil {
    return nil, err
}

// ...
}

```

Fungsi tersebut menerima 3 buah parameter.

- Parameter `url`, adalah alamat tujuan request.
- Parameter `method`, bisa GET, POST, PUT, ataupun method valid lainnya sesuai spesifikasi [RFC 2616](#).
- Parameter `data`, isinya boleh kosong. Jika ada isinya, data tersebut di-encode ke bentuk JSON untuk selanjutnya disisipkan pada body request.

Buat objek request lewat `http.NewRequest()`. Sisipkan ke-3 parameter tersebut.

Selanjutnya buat objek client. Dari client ini request di-trigger, menghasilkan objek response.

```

client := new(http.Client)

response, err := client.Do(request)
if response != nil {
    defer response.Body.Close()
}
if err != nil {
    return nil, err
}

responseBody := make(M)
err = json.NewDecoder(response.Body).Decode(&responseBody)
if err != nil {
    return nil, err
}

return responseBody, nil

```

Decode response tersebut ke tipe `M`, lalu tampilkan hasilnya.

Buat fungsi `main()`. Panggil fungsi `doRequest()` yang sudah dibuat. Untuk payload silakan isi sesuka hati, asalkan ada item dengan key `Name`. Lalu tampilkan response body hasil pemanggilan fungsi `doRequest()`.

```

func main() {
    baseURL := "http://localhost:9000"
    method := "POST"
    data := M{"Name": "Noval Agung"}

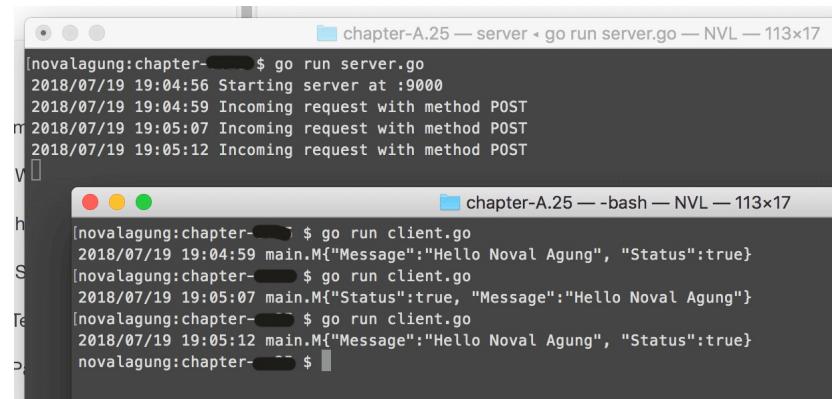
    responseBody, err := doRequest(baseURL+ "/data", method, data)
    if err != nil {
        log.Println("ERROR", err.Error())
        return
    }

    log.Printf("%#v \n", responseBody)
}

```

### C.24.3. Testing

Jalankan aplikasi server, buka prompt terminal/CMD baru, lalu jalankan aplikasi client.



```
[novalagung:chapter-] $ go run server.go
2018/07/19 19:04:56 Starting server at :9000
2018/07/19 19:04:59 Incoming request with method POST
2018/07/19 19:05:07 Incoming request with method POST
2018/07/19 19:05:12 Incoming request with method POST
[novalagung:chapter-] $ go run client.go
2018/07/19 19:04:59 main.M{"Message":"Hello Noval Agung", "Status":true}
[novalagung:chapter-] $ go run client.go
2018/07/19 19:05:07 main.M{"Status":true, "Message":"Hello Noval Agung"}
[novalagung:chapter-] $ go run client.go
2018/07/19 19:05:12 main.M{"Message":"Hello Noval Agung", "Status":true}
[novalagung:chapter-] $
```

## C.25. Secure & Insecure Client HTTP Request

Pada bab ini topik yang dibahas adalah cara melakukan http request ke SSL/TLS-enabled web server, menggunakan dua teknik:

- Insecure request
- Secure request menggunakan file certificate

### C.25.1. Handshake

Sebelum masuk ke inti pembahasan, kita perlu mempelajari terlebih dahulu tentang pembeda antara secure request dan http request biasa.

Dalam secure request, sebelum data benar-benar diterima oleh server, terjadi proses negosiasi antara client (yg men-dispatch request) dan server (tujuan request), proses ini biasa disebut dengan handshake.

Proses negosiasi tersebut dipecah menjadi 5 fase.

1. Fase **Client Hello**. Pada fase ini handshake dimulai dengan client mengirimkan pesan yang kita sebut dengan **client hello** ke se server. Pesan tersebut berisikan semua informasi milik client yang diperlukan oleh server untuk bisa terhubung dengan client via SSL. Informasi yang dimaksud diantaranya adalah versi SSL/TLS dan konfigurasi cipher. Cipher suite sendiri adalah seperangkat algoritma, digunakan untuk membantu pengamanan koneksi yang menerapkan TLS/SSL.
2. Fase **Server Hello**. Setelah diterima, server merespon dengan pesan yang mirip, yaitu **server hello**, isinya juga informasi yang kurang lebih sejenis. Informasi ini diperlukan oleh client untuk bisa terhubung balik dengan server.
3. Fase **Otentikasi dan Pre-Master Secret**. Setelah kontak antara client dan server terjadi, server mengenalkan dirinya ke client lewat file certificate. Anggap saja certificate tersebut sebagai KTP (Kartu Tanda Penduduk). Client selanjutnya melakukan pengecekan, apakah KTP tersebut valid dan dipercaya, atau tidak. Jika memang terpercaya, client selanjutnya membuat data yang disebut dengan **pre-master secret**, meng-enkripsi-nya menggunakan public key, lalu mengirimnya ke server sebagai response.
4. Fase **Decryption and Master Secret**. Data encrypted pre-master secret yang dikirim oleh client diterima oleh server. Data tersebut kemudian di-decrypt menggunakan private key. Selanjutnya server dan client melakukan beberapa hal untuk men-generate **master secret** lewat cipher yang sudah disepakati.
5. Fase **Encryption with Session Key**. Server dan client melakukan pertukaran pesan untuk menginfokan bahwa data yang dikirim dalam request tersebut dan request-request selanjutnya akan di-enkripsi.

Sekarang kita tau, bahwa agar komunikasi antara client dan server bisa terjalin, pada sisi client harus ada file certificate, dan pada sisi server harus private key & certificate.

OK, saya rasa bagian teori sudah cukup, mari kita lanjut ke bagian praktek.

### C.25.2. Persiapan

Salin projek pada bab sebelumnya, bab [A25 - Client HTTP Request](#) sebagai folder projek baru.

### C.25.3. Konfigurasi SSL/TLS pada Web Server

Di bab A25 kita sudah belajar implementasi client http request, penerapannya dengan 2 buah aplikasi terpisah, satu aplikasi web server dan satu lagi adalah aplikasi consumer.

Kita perlu menambahkan sedikit modifikasi pada aplikasi web server (yang sudah di salin), mengaktifkan SSL/TLS-nya dengan cara mengubah bagian `.ListenAndServe()` menjadi `.ListenAndServeTLS()`, dengan disisipkan dua parameter berisi path certificate dan private key.

```
err := server.ListenAndServeTLS("server.crt", "server.key")
```

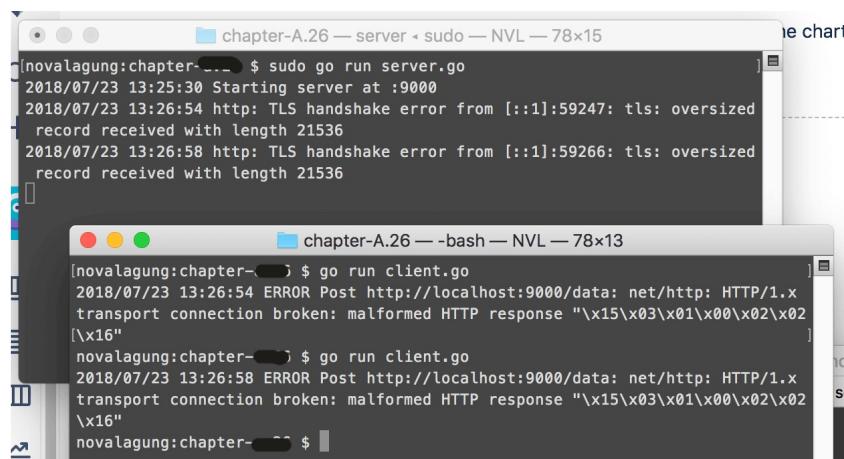
Silakan generate certificate dan private key baru, caranya sama seperti pada bab [A23 - HTTPS/TLS Web Server](#).

Konfigurasi SSL/TLS lewat `server.ListenAndServeTLS("server.crt", "server.key")` merupakan cara yang paling mudah dengan konfigurasi adalah paling minimal.

## C.25.4. Insecure Request

Dari yang sudah dijelaskan di atas, agar komunikasi antara client dan server bisa ter-enkripsi, di sisi client atau consumer harus ada yang namanya file certificate.

Jika client tidak menyertakan certificate dalam request-nya, maka pasti terjadi error (pada saat handshake). Contohnya bisa dilihat pada screenshot berikut.



Akan tetapi, jika memang client tidak memiliki certificate dan komunikasi ingin tetap dilakukan, masih bisa (dengan catatan server meng-allow kapabilitas ini), caranya yaitu menggunakan teknik *insecure request*.

Dalam insecure request, komunikasi terjalin tanpa ada proses enkripsi data.

Cara membuat insecure request sangat mudah, cukup aktifkan atribut `insecure` pada request. Misal menggunakan `curl`, maka cukup tambahkan flag `--insecure` pada command.

```
curl -X POST https://localhost/data \
--insecure \
-H 'Content-Type: application/json' \
-d '{"Name": "Noval Agung"}'
```

Penerapan `insecure request` dalam goLang juga tidak terlalu sulit. Pada object `http.Client`, isi property `.Transport` dengan objek baru buatan struct `http.Transport` yang didalamnya berisi konfigurasi `insecure request`.

```
client := new(http.Client)
client.Transport = &http.Transport{
    TLSClientConfig: &tls.Config{InsecureSkipVerify: true},
}
```

Ubah kode pada aplikasi client (yang sudah disalin) seperti di atas. Jangan lupa juga untuk mengganti protokol base url destinasi, dari `http` ke `https`.

```
baseURL := "https://localhost:9000"
```

Jalankan ulang aplikasi server yang sudah ssl-enabled dan aplikasi client yang sudah dikonfigurasi untuk insecure request, lalu test hasilnya.

```
chapter-A.26 — server — NVL — 68x16
[novalagung:chapter-] $ sudo go run server.go
[Password:
2018/07/23 13:19:34 Starting server at
2018/07/23 13:19:40 Incoming request with method POST
2018/07/23 13:19:53 Incoming request with method POST

chapter-A.26 — bash — NVL — 82x14
[novalagung:chapter-] $ go run client-insecure.go
2018/07/23 13:19:40 main.M{"Message":"Hello Noval Agung", "Status":true}
[novalagung:chapter-] $ go run client-insecure.go
2018/07/23 13:19:53 main.M{"Message":"Hello Noval Agung", "Status":true}
novalagung:chapter-]
```

## C.25.5. Secure Request

Secure request adalah bentuk request yang datanya ter-enkripsi, bisa dibilang kebalikan dari insecure request. Request jenis ini pada sisi client atau consumer membutuhkan konfigurasi dimana file certificate diperlukan.

Secure request bisa dilakukan dengan mudah di golang. Mari langsung saja kita praktikan. Pertama, pada file consumer, tambahkan package `crypto/x509`.

```
import (
    // ...
    "crypto/x509"
)
```

X.509 adalah standar format public key certificates.

Lalu buat objek baru bertipe `x509.CertPool` lewat `x509.NewCertPool()`. Objek ini nantinya menampung list certificate yang digunakan.

Buat objek menggunakan struct `tls.Config`, dengan isi property `RootCAs` adalah objek list certificate yang sudah dibuat.

Isi `client.Transport` dengan konfigurasi secure request. Hapus saja konfigurasi insecure request sebelumnya.

Kurang lebih kode-nya seperti berikut.

```
certFile, err := ioutil.ReadFile("server.crt")
if err != nil {
    return nil, err
}

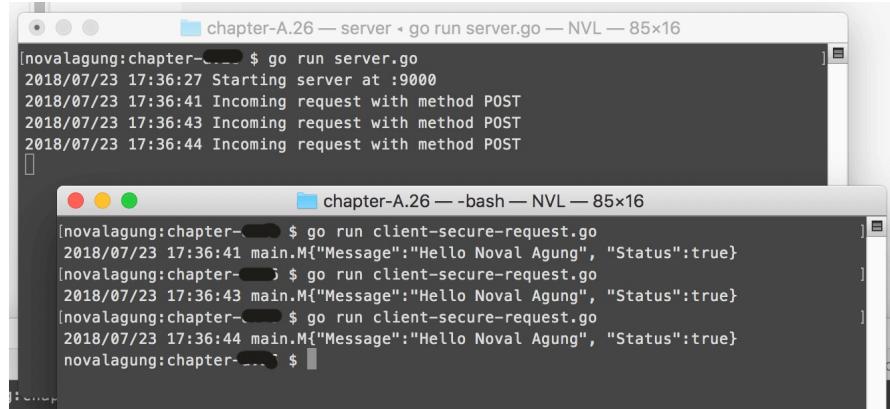
caCertPool := x509.NewCertPool()
caCertPool.AppendCertsFromPEM(certFile)

tlsConfig := &tls.Config{RootCAs: caCertPool}
tlsConfig.BuildNameToCertificate()
```

```
client := new(http.Client)
client.Transport = &http.Transport{
    TLSClientConfig: tlsConfig,
}
```

Bisa dilihat pada kode di atas, file `server.crt` dibaca isinya, lalu dimasukan ke `caCertPool`. Objek `caCertPool` ini bisa menampung banyak certificate, jika memang dibutuhkan banyak.

OK, silakan langsung run aplikasi untuk testing.



## C.25.6. Konfigurasi SSL/TLS Lanjutan

Di atas kita sudah belajar cara setting SSL/TLS pada web server, dengan konfigurasi minimal menggunakan

```
server.ListenAndServeTLS("server.crt", "server.key") .
```

Konfigurasi yang lebih complex bisa kita lakukan menggunakan `tls.Config`. Buat objek menggunakan struct tersebut lalu manfaatkan property struct-nya untuk menciptakan konfigurasi yang sesuai dengan kebutuhan. Contoh kurang lebih seperti kode di bawah ini.

```
certPair1, err := tls.LoadX509KeyPair("server.crt", "server.key")
if err != nil {
    log.Fatalf("Failed to start web server", err)
}

tlsConfig := new(tls.Config)
tlsConfig.NextProtos = []string{"http/1.1"}
tlsConfig.MinVersion = tls.VersionTLS12
tlsConfig.PreferServerCipherSuites = true

tlsConfig.Certificates = []tls.Certificate{
    certPair1, /* add other certificates here */
}
tlsConfig.BuildNameToCertificate()

tlsConfig.ClientAuth = tls.VerifyClientCertIfGiven
tlsConfig.CurvePreferences = []tls.CurveID{
    tls.CurveP521,
    tls.CurveP384,
    tls.CurveP256,
}
tlsConfig.CipherSuites = []uint16{
    tls.TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,
    tls.TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,
}
```

Tampung saja objek cetakan `server.TLSConfig` di atas ke dalam `server.TLSConfig`. Jika file certificate dan private key sudah ditambahkan dalam `tlsConfig`, maka dalam pemanggilan `server.ListenAndServeTLS()` kosongkan saja parameter-nya.

```
server := new(http.Server)
server.Handler = mux
server.Addr = ":9000"
server.TLSConfig = tlsConfig

err := server.ListenAndServeTLS("", "")
if err != nil {
    log.Fatalln("Failed to start web server", err)
}
```

Tujuan mengapa penulis tambahkan sub bab **Konfigurasi SSL/TLS Lanjutan** ini adalah agar pembaca tau bahwa konfigurasi SSL/TLS yang kompleks bisa dilakukan dengan mudah dalam aplikasi web go lang. Mengenai pembahasan tiap-tiap property silakan pelajari sendiri.

## C.26. FTP

Pada bab ini kita akan belajar cara melakukan pertukaran data lewat FTP (File Transfer Protocol) menggunakan GoLang.

Definisi mengenai FTP sendiri adalah sebuah protokol network standar yang digunakan untuk pertukaran atau transfer data antar client dan server.

Sebelum memulai, ada satu hal penting yang perlu dipersiapkan, yaitu sebuah server dengan FTP server ter-install. Jika tidak ada, bisa menggunakan library `ftp` untuk set up ftp server di local (untuk keperluan belajar).

Dalam server tersebut, siapkan beberapa file dan folder dengan struktur sebagai berikut.

```
i  C:\Users\novalagung\Desktop\downloaded\mov\cc\mp4\ } novalagung:folder $ tree
.
├── movie.mp4
└── somefolder
    └── test3.txt
}   ├── test1.txt
f   └── test2.txt
f
f
i.e err != nil f
1 directory, 4 files
novalagung:folder $
```

- File `test1.txt`, isi dengan apapun.
- File `test2.txt`, isi dengan apapun.
- File `somefolder/test3.txt`, isi dengan apapun.
- File `movie.mp4`, gunakan file seadanya.

Library FTP client yang kita gunakan adalah [github.com/jlaffaye/ftp](https://github.com/jlaffaye/ftp).

### C.26.1. Koneksi ke Server

Buat satu buah folder projek baru dengan isi `main.go`. Di dalam file main akan kita isi dengan beberapa operasi FTP seperti upload, download, akses ke direktori dan lainnya.

OK, langsung saja, silakan tulis kode berikut.

```
package main

import (
    "fmt"
    "github.com/jlaffaye/ftp"
    "log"
)

func main() {
    const FTP_ADDR = "0.0.0.0:2121"
    const FTP_USERNAME = "admin"
    const FTP_PASSWORD = "123456"

    // ...
}
```

Tiga buah konstanta dengan prefix `FTP_` disiapkan, isinya adalah credentials FTP untuk bisa melakukan koneksi FTP ke server.

Di dalam `main()`, tambahkan kode untuk terhubung dengan server.

```
conn, err := ftp.Dial(FTP_ADDR)
if err != nil {
    log.Fatal(err.Error())
}

err = conn.Login(FTP_USERNAME, FTP_PASSWORD)
if err != nil {
    log.Fatal(err.Error())
}
```

Cara koneksi ke server melalui FTP dipecah menjadi dua tahap. Pertama adalah menggunakan `ftp.Dial()` dengan argumen adalah alamat server (beserta port). Statement tersebut mengembalikan objek bertipe `*ftp.ServerConn` yang ditampung oleh variabel `conn`.

Tahap kedua, lewat objek `conn`, panggil method `.Login()` dengan disisipi argumen username dan password FTP.

## C.26.2. Menampilkan Semua File Menggunakan `.List()`

Lewat tipe `*ftp.ServerConn`, semua method untuk operasi FTP bisa diakses. Salah satu dari method tersebut adalah `.List()`, gunanya untuk listing semua file yang ada di server. Operasi ini sama dengan `ls`.

```
fmt.Println("===== PATH ./")

entries, err := conn.List(".")
if err != nil {
    log.Fatal(err.Error())
}
for _, entry := range entries {
    fmt.Println(" ->", entry.Name, getStringEntryType(entry.Type))
}
```

Method `.List()` di atas dipanggil dengan argumen adalah string `"."`, berarti semua asset dalam `"."` atau **current path** dimunculkan.

Method tersebut mengembalikan slice yang tiap elemennya adalah representasi tiap file. Pada kode di atas, semua file dimunculkan ke console, nama dan tipe-nya.

Property `.Type` milik `entry` tipenya adalah `ftp.EntryType`, yang sebenarnya `int`. Fungsi `getStringEntryType()` kita buat untuk menampilkan keterangan dalam string sesuai dengan tipe.

```
func getStringEntryType(t ftp.EntryType) string {
    switch t {
    case ftp.EntryTypeFile:
        return "(file)"
    case ftp.EntryTypeFolder:
        return "(folder)"
    case ftp.EntryTypeLink:
        return "(link)"
    default:
        return ""
    }
}
```

Jalankan aplikasi lihat hasilnya.

```
[novalagung:folder $ tree
.
├── movie.mp4
└── somefolder
    └── test3.txt
test1.txt
test2.txt
1 directory, 4 files
novalagung:folder $]

[novalagung:chapter-b.22 $ go run main.go
===== PATH ./
-> movie.mp4 (file)
-> somefolder (folder)
-> test1.txt (file)
-> test2.txt (file)
novalagung:chapter-b.22 $]
```

Jika dibandingkan dengan file yang ada di server, ada satu yang tidak muncul, yaitu `somefolder/test3.txt`. Hal ini dikarenakan file yang di-list adalah yang ada pada `"."` atau **current path**. File `test3.txt` berada di dalam sub folder `somefolder`.

### C.26.3. Pindah Ke Folder Tertentu Menggunakan `.ChangeDir()`

Selanjutnya, kita akan coba masuk ke folder `somefolder`, lalu menampilkan isinya. Gunakan method `.ChangeDir()`, sisipkan path folder tujuan sebagai argument.

```
fmt.Println("===== PATH ./somefolder")

err = conn.ChangeDir("./somefolder")
if err != nil {
    log.Fatal(err.Error())
}
```

Setelah itu, list semua file. Tetap gunakan `"."` sebagai argument pemanggilan method `.List()` untuk listing current path. Karena kita sudah masuk ke folder `./somefolder`, maka path tersebutlah yang menjadi current path.

```
entries, err = conn.List(".")
if err != nil {
    log.Fatal(err.Error())
}
for _, entry := range entries {
    fmt.Println(" ->", entry.Name, getStringEntryType(entry.Type))
}
```

Jalankan aplikasi, lihat lagi hasilnya.

```
[novalagung:folder $ tree
.
├── movie.mp4
└── somefolder
    └── test3.txt
test1.txt
test2.txt
1 directory, 4 files
novalagung:folder $]

[novalagung:chapter-b.22 $ go run main.go
===== PATH ./
-> movie.mp4 (file)
-> somefolder (folder)
-> test1.txt (file)
-> test2.txt (file)
===== PATH ./somefolder
-> test3.txt (file)
novalagung:chapter-b.22 $]
```

## C.26.4. Pindah Ke Parent Folder Menggunakan .ChangeDirToParent()

Gunakan method `.ChangeDirToParent()` untuk mengubah aktif path ke parent path. Tambahkan kode berikut, agar current path `./somefolder` kembali menjadi `..`.

```
err = conn.ChangeDirToParent()
if err != nil {
    log.Fatal(err.Error())
}
```

## C.26.5. Mengambil File Menggunakan `.Retr()` Lalu Membaca Isinya

Cara mengambil file adalah dengan method `.Retr()`. Tulis saja path file yang ingin diambil sebagai argumen. Nilai baliknya adalah objek bertipe `*ftp.Response` dan error (jika ada).

```
fmt.Println("===== SHOW CONTENT OF FILES")

fileTest1Path := "test1.txt"
fileTest1, err := conn.Retr(fileTest1Path)
if err != nil {
    log.Fatal(err.Error())
}

test1ContentInBytes, err := ioutil.ReadAll(fileTest1)
fileTest1.Close()
if err != nil {
    log.Fatal(err.Error())
}

fmt.Println(" ->", fileTest1Path, "->", string(test1ContentInBytes))
```

Baca isi objek response tersebut menggunakan method `.Read()` miliknya, atau bisa juga menggunakan `ioutil.ReadAll()` lebih praktisnya (nilai baliknya bertipe `[]byte` maka cast ke tipe `string` terlebih dahulu untuk menampilkan isinya).

Jangan lupa untuk import package `io/ioutil`.

Di kode di atas file `test1.txt` dibaca. Lakukan operasi yang sama pada file `somefolder/test3.txt`.

```
fileTest2Path := "somefolder/test3.txt"
fileTest2, err := conn.Retr(fileTest2Path)
if err != nil {
    log.Fatal(err.Error())
}

test2ContentInBytes, err := ioutil.ReadAll(fileTest2)
fileTest2.Close()
if err != nil {
    log.Fatal(err.Error())
}

fmt.Println(" ->", fileTest2Path, "->", string(test2ContentInBytes))
```

Jangan lupa juga untuk meng-close objek bertipe `*ftp.Response`, setelah dibaca isinya.

Jalankan aplikasi, cek hasilnya.

The screenshot shows two terminal windows side-by-side. The left window, titled 'novalagung:folder', contains the command 'cat test1.txt' followed by its output: 'hello world'. The right window, titled 'novalagung:chapter-B.22', contains the command 'go run main.go' followed by its output. The output includes a list of files and their paths, and then a section labeled '===== SHOW CONTENT OF FILES' which shows the contents of 'test1.txt' as 'hello world' and 'somefolder/test3.txt' as 'threeeee'. Both outputs have specific lines highlighted with blue boxes.

```
[novalagung:folder $ cat test1.txt
[hello world
[novalagung:folder $ cat somefolder/test3.txt
[threeeee
novalagung:folder $ ]]

[novalagung:chapter-B.22 $ go run main.go
===== PATH ./
-> movie.mp4 (file)
-> somefolder (folder)
-> test1.txt (file)
-> test2.txt (file)
===== PATH ./somefolder
■ 179   file
■ 180   if
■ 181   ===== SHOW CONTENT OF FILES
■ 182   }   -> test1.txt -> [hello world
■ 183   }   -> somefolder/test3.txt -> [threeeee
■ 184   fmt
■ 185   ```

novalagung:chapter-B.22 $ ]]
```

Bisa dilihat, isi file yang dibaca adalah aslinya.

## C.26.6. Download File

Proses download secara teknis adalah memindahkan **isi file** dari remote server ke local server. Di local, dibuatkan sebuah file yang nantinya akan menampung isi file dari remote server yang di-transfer. Seberapa cepat proses download berlangsung sangat tergantung kepada besar file yang isinya sedang di transfer (dan beberapa faktor lainnya).

Di golang, penerapan download lewat FTP kurang lebih adalah sama. perlu dibuat terlebih dahulu file di local untuk menampung isi file yang diambil dari remote server.

OK, mari kita praktikan. File `movie.mp4` yang berada di server akan kita unduh ke local.

```
fmt.Println("===== DOWNLOAD FILE TO LOCAL")

fileMoviePath := "movie.mp4"
fileMovie, err := conn.Retr(fileMoviePath)
if err != nil {
    log.Fatal(err.Error())
}

destinationMoviePath := "/Users/novalagung/Desktop/downloaded-movie.mp4"
f, err := os.Create(destinationMoviePath)
if err != nil {
    log.Fatal(err.Error())
}

_, err = io.Copy(f, fileMovie)
if err != nil {
    log.Fatal(err.Error())
}
fileMovie.Close()
f.Close()

fmt.Println("File downloaded to", destinationMoviePath)
```

Pertama ambil file tujuan menggunakan `.Retr()`. Lalu buat file di local. Pada contoh di atas nama file di local adalah berbeda dengan nama file asli, `downloaded-movie.mp4`. Buat file tersebut menggunakan `os.Create()`.

Setelah itu, copy isi file yang sudah diambil dari server, ke `downloaded-movie.mp4` menggunakan `io.Copy()`. Setelah proses transfer selesai, jangan lupa untuk close file dari remote dan juga file di local.

Operasi di atas membutuhkan dua package lain, yaitu `io` dan `os`, maka import kedua package tersebut.

Jalankan aplikasi, coba cek md5 sum dari kedua file, harusnya sama.

```

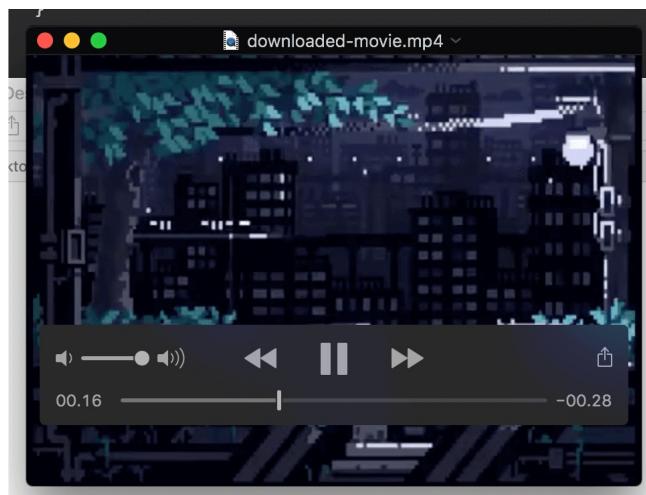
117 ——— fileMovie,err := client.Retr(fileMoviePath)
      folder —— bash — NVL — 72x13
[novalagung:folder $] md5 movie.mp4
MD5 (movie.mp4) = f30552d35fad2e7d908bf2727a4b5126
[novalagung:folder $]

      chapter-B.22 —— bash — NVL — 95x18
[novalagung:chapter-b.22 $] go run main.go
===== PATH ./
-> movie.mp4 (file)
-> somefolder (folder)
-> test1.txt (file)
-> test2.txt (file)
===== PATH ./somefolder
-> test3.txt (file)
===== SHOW CONTENT OF FILES
-> test1.txt -> hello world
-> somefolder/test3.txt -> threeeee
===== DOWNLOAD FILE TO LOCAL
File downloaded to /Users/novalagung/Desktop/downloaded-movie.mp4
[novalagung:chapter-b.22 $]
[novalagung:chapter-b.22 $] md5 /Users/novalagung/Desktop/downloaded-movie.mp4
MD5 (/Users/novalagung/Desktop/downloaded-movie.mp4) = f30552d35fad2e7d908bf2727a4b5126
[novalagung:chapter-b.22 $]

      145 // err = client.Stor("./somefolder/logo.png", f)

```

Coba buka `downloaded-movie.mp4`, jika proses transfer sukses maka pasti bisa dibuka.



## C.26.7. Upload File

Upload file adalah kebalikan dari download file. File dari lokal ditransfer ke server. Mari langsung kita praktikan.

Pertama buka file tujuan menggunakan `os.open()`. Lalu panggil method `.Store()` milik `conn`, sisipkan path file tujuan remote server sebagai parameter pertama, lalu objek file di local sebagai parameter kedua.

```

fmt.Println("===== UPLOAD FILE TO FTP SERVER")

sourceFile := "/Users/novalagung/Desktop/Go-Logo_Aqua.png"
f, err = os.Open(sourceFile)
if err != nil {
    log.Fatal(err.Error())
}

destinationFile := "./somefolder/logo.png"
err = conn.Store(destinationFile, f)
if err != nil {
    log.Fatal(err.Error())
}

```

```
f.Close()

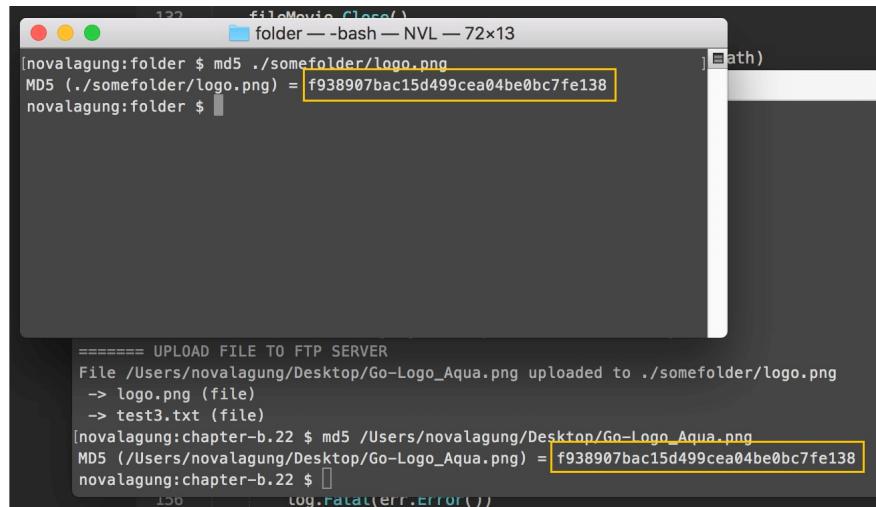
fmt.Println("File", sourceFile, "uploaded to", destinationFile)
```

File `Go-Logo_Aqua.png` di upload ke server pada path `./somefolder/logo.png`. Coba list untuk mengecek apakah file sudah terupload.

```
entries, err = conn.List("./somefolder")
if err != nil {
    log.Fatal(err.Error())
}

for _, entry := range entries {
    fmt.Println(" ->", entry.Name, getStringEntryType(entry.Type))
}
```

Jalankan aplikasi, cek hasilnya. Untuk memvalidasi bahwa file di client dan di server sama, bandingkan md5 sum kedua file.



The screenshot shows a terminal window with two sessions. The top session is in a folder named 'somefolder' and shows the command `md5 ./somefolder/logo.png` being run, with the output `MD5 (. /somefolder/logo.png) = f938907bac15d499cea04be0bc7fe138`. The bottom session is in a folder named 'chapter-b.22' and shows the command `md5 /Users/novalagung/Desktop/Go-Logo_Aqua.png` being run, with the output `MD5 (/Users/novalagung/Desktop/Go-Logo_Aqua.png) = f938907bac15d499cea04be0bc7fe138`. Both outputs are highlighted with yellow boxes.

```
fileMovie_Close()
[novalagung:folder $ md5 ./somefolder/logo.png
MD5 (. /somefolder/logo.png) = f938907bac15d499cea04be0bc7fe138
novalagung:folder $ ] math

===== UPLOAD FILE TO FTP SERVER
File /Users/novalagung/Desktop/Go-Logo_Aqua.png uploaded to ./somefolder/logo.png
-> logo.png (file)
-> test3.txt (file)
[novalagung:chapter-b.22 $ md5 /Users/novalagung/Desktop/Go-Logo_Aqua.png
MD5 (/Users/novalagung/Desktop/Go-Logo_Aqua.png) = f938907bac15d499cea04be0bc7fe138
novalagung:chapter-b.22 $ ] log.Fatal(err.Error())
```

- [ftpd](#), by Lunny Xiao
- [ftp](#), by Julien Laffaye, ISC license

## C.27. SSH & SFTP

Pada bab ini kita akan belajar cara untuk me-remote server lewat protokol SSH (Secure Shell).

Protokol SSH digunakan untuk melakukan remote login secara aman/secure ke server tujuan. Komunikasi yang terjadi lewat SSH di-encrypt sehingga aman.

Golang menyediakan package [golang.org/x/crypto/ssh](https://golang.org/x/crypto/ssh), berisi cukup banyak API untuk keperluan operasi yang berhubungan dengan protokol SSH.

### C.27.1. Otentikasi SSH

Buat folder projek baru, isinya file `main.go` , didalamnya tulis kode berikut.

```
package main

import (
    "golang.org/x/crypto/ssh"
    "io/ioutil"
    "log"
    "os"
)

func main() {
    const SSH_ADDRESS = "0.0.0.0:22"
    const SSH_USERNAME = "user"
    const SSH_PASSWORD = "password"

    sshConfig := &ssh.ClientConfig{
        User:           SSH_USERNAME,
        HostKeyCallback: ssh.InsecureIgnoreHostKey(),
        Auth: []ssh.AuthMethod{
            ssh.Password(SSH_PASSWORD),
        },
    }

    // ...
}
```

Variabel `sshConfig` di atas adalah objek pointer cetakan struct `ssh.ClientConfig`. Objek bertipe ini nantinya digunakan untuk otentikasi SSH.

Pada kode di atas, tiga buah konstanta dengan prefix `SSH_*` disiapkan. Credentials `username` dan `password` disisipkan sebagai property objek `sshConfig`. Bisa dilihat pada property `Auth` , isinya adalah slice `ssh.AuthMethod` dengan satu buah element yaitu `ssh.Password()` . Konfigurasi seperti ini dipakai jika **otentifikasi menggunakan username dan password**.

Jika otentikasi dilakukan menggunakan **identity file**, maka gunakan kode berikut.

```
const SSH_ADDRESS = "192.168.0.24:22"
const SSH_USERNAME = "user"
const SSH_KEY = "path/to/file/identity.pem"

sshConfig := &ssh.ClientConfig{
    User:           SSH_USERNAME,
    HostKeyCallback: ssh.InsecureIgnoreHostKey(),
    Auth: []ssh.AuthMethod{
        PublicKeyFile(SSH_KEY),
    }
}
```

```

    },
}

func PublicKeyFile(file string) ssh.AuthMethod {
    buffer, err := ioutil.ReadFile(file)
    if err != nil {
        return nil
    }

    key, err := ssh.ParsePrivateKey(buffer)
    if err != nil {
        return nil
    }

    return ssh.PublicKeys(key)
}

```

Silakan pilih jenis otentikasi sesuai dengan yang di dukung oleh remote server.

Selanjutnya, dalam fungsi `main()`, buat koneksi baru ke server tujuan menggunakan `ssh.Dial()`. Statement ini mengembalikan objek `*ssh.Client`, pemanggilannya sendiri membutuhkan 3 parameter.

- Isi argument pertama isi dengan `"tcp"`, hal ini dikarenakan protokol tersebut digunakan dalam SSH.
- Argument ke-2 diisi dengan alamat tujuan server.
- Argument ke-3 diisi dengan objek `sshConfig`.

```

// ...

client, err := ssh.Dial("tcp", SSH_ADDRESS, sshConfig)
if client != nil {
    defer client.Close()
}
if err != nil {
    log.Fatal("Failed to dial. " + err.Error())
}

```

## C.27.2. Session & Run Command

Dari objek `client`, buat session baru, caranya dengan mengakses method `.NewSession()`.

```

session, err := client.NewSession()
if session != nil {
    defer session.Close()
}
if err != nil {
    log.Fatal("Failed to create session. " + err.Error())
}

```

Pada session, set tiga koneksi standar IO: `stdin`, `stdout`, dan `stderr`; ke standard IO sistem operasi.

```

session.Stdin = os.Stdin
session.Stdout = os.Stdout
session.Stderr = os.Stderr

```

OK, semua persiapan sudah cukup. Sekarang coba jalankan salah satu command seperti `ls`.

```

err = session.Run("ls -l ~/")
if err != nil {
    log.Fatal("Command execution error. " + err.Error())
}

```

Jalankan aplikasi untuk mengetes hasilnya.

```
total 160
drwxr-x---+ 5 novalagung staff 160 Jan 11 2016 Public
drwxr-xr-x 155 novalagung staff 4960 Jul 9 2017 node_modules
drwxr-xr-x 4 novalagung staff 128 Sep 17 2017 itsm
drwxr-xr-x 4 novalagung staff 128 Oct 21 2017 Oracle
-rw-r--r-- 1 novalagung staff 103 Mar 6 12:38 java1.log
-rw-r--r-- 1 novalagung staff 103 Apr 27 17:18 java0.log
drwxr-xr-x 4 novalagung staff 128 Apr 29 18:27 GitBook
```

### C.27.3. Penggunaan `session.StdinPipe()` untuk Run Multiple Command

Ada beberapa cara yang bisa digunakan untuk menjalankan banyak command via SSH, cara paling mudah adalah dengan menggabung commands dengan operator `&&`.

Alternatif metode lainnya, bisa dengan memanfaatkan `StdinPipe` milik session. Cukup tulis command yang diinginkan ke objek stdin pipe tersebut.

Method `.StdinPipe()` mengembalikan objek stdin pipe yang tipenya adalah `io.WriteCloser`. Tipe ini merupakan gabungan dari `io.Writer`, `io.Reader`, dan `io.Closer`.

Mari kita praktikan, salin kode sebelumnya ke file baru, hapus semua baris kode setelah statement pembuatan session.

Siapkan variabel untuk menampung objek default stdin pipe milik session.

```
var stdout, stderr bytes.Buffer
session.Stdout = &stdout
session.Stderr = &stderr

stdin, err := session.StdinPipe()
if err != nil {
    log.Fatal("Error getting stdin pipe. " + err.Error())
}
```

Jalankan command prompt di remote host, menggunakan perintah `.start()`. Pilih unix shell yang diinginkan dengan menuliskannya sebagai argument pemanggilan method `.start()`. Pada tutorial ini kita menggunakan unix shell `bash`.

```
err = session.Start("/bin/bash")
if err != nil {
    log.Fatal("Error starting bash. " + err.Error())
}
```

Method `.start()` dan `.Run()` memiliki kesamaan dan perbedaan. Salah satu kesamaannya adalah kedua method tersebut digunakan untuk menjalankan perintah shell, yang eksekusinya akan selesai ketika perintah yang dijalankan selesai, contoh: `.Start("ls -l ~")` dan `.Run("ls -l ~")`, kedua statement tersebut menghasilkan proses dan output yang sama.

Sedangkan perbedaannya, method `.Start()` bisa digunakan untuk memilih command line interpreter atau shell (pada contoh ini `bash`), lalu memanfaatkannya untuk eksekusi banyak shell command dengan cara dilewatkan ke stdin pipe.

Pemanggilan method `.Start()` dan `.Run()` hanya bisa dilakukan sekali untuk tiap session.

Selanjutnya, siapkan commands dalam slice, untuk mempermudah eksekusinya. Lakukan perulangan pada slice tersebut, lalu tulis command ke dalam objek stdin pipe. Pastikan command terakhir yang dieksekusi adalah `exit` untuk mengakhiri shell.

```
commands := []string{
    "cd /where/is/the/gopath",
    "cd src/myproject",
    "ls",
    "exit",
}
for _, cmd := range commands {
    if _, err = fmt.Fprintln(stdin, cmd); err != nil {
        log.Fatal(err)
    }
}
```

Statement `fmt.Fprintln()` digunakan untuk menuliskan sesuatu ke objek `io.Writer`. Objek stdin pipe kita sisipkan sebagai parameter pertama, lalu shell command sebagai parameter setelahnya.

Selain `fmt.Fprintln()`, ada juga `fmt.Println()` dan `fmt.Sprintf()`.

Statement yang paling sering kita gunakan, yaitu `fmt.Print()`, isinya sebenarnya memanggil `fmt.Fprint()`, dengan parameter pertama `io.Writer` diisi dengan `os.Stdout`.

Gunakan method `.Wait()` milik `session` untuk menunggu banyak command yang dieksekusi selesai terlebih dahulu. Kemudian tangkap output `stdout` dan `stderr`nya lalu tampilkan.

```
err = session.Wait()
if err != nil {
    log.Fatal(err)
}

outputErr := stderr.String()
fmt.Println("===== ERROR")
fmt.Println(strings.TrimSpace(outputErr))

outputString := stdout.String()
fmt.Println("===== OUTPUT")
fmt.Println(strings.TrimSpace(outputString))
```

Jalankan aplikasi untuk melihat hasilnya.

```
novalagung:chapter-b.27 $ go run 2-multiple-commands.go
=====
ERROR

=====
OUTPUT
assets
backup-db
client
commit.sh
db
nohup.out
package-lock.json
pdfer
README.md
server
vendor
widget-gantt-toolset
novalagung:chapter-b.27 $
```

Output dalam banyak command muncul setelah semua command berhasil dieksekusi. Statement `session.Wait()` adalah blocking.

Jika ingin eksekusi command dan pengambilan outpunya tidak blocking, manfaatkan `.StdoutPipe()`, `.StderrPipe()`, dan goroutine untuk pengambilan output hasil eksekusi command.

## C.27.4. Transfer File via SFTP

Transfer file antara client dan server bisa dilakukan lewat protokol SSH, dengan memanfaatkan SFTP (SSH File Transfer Protocol). Penerapannya sebenarnya bisa dilakukan cukup menggunakan API yang disediakan oleh package `golang.org/x/crypto/ssh`, namun pada bagian ini kita akan menggunakan 3rd party library lain untuk mempermudah penerapannya. Library tersebut adalah [github.com/pkg/sftp](https://github.com/pkg/sftp).

Mari kita praktikan, salin kode sebelumnya ke file baru, hapus semua baris kode setelah statement pembuatan client. Kemudian, go get package tersebut, lalu import.

Buat objek sftp client, objek ini merupakan superset dari objek ssh client.

```
sftpClient, err := sftp.NewClient(client)
if err != nil {
    log.Fatal("Failed create client sftp client. " + err.Error())
}
```

Kita akan menggunakan sample file di lokal untuk ditransfer ke server. Mekanisme-nya sama seperti pada transfer file via ftp, yaitu dengan menyiapkan file kosong di sisi server, lalu meng-copy isi file di lokal ke file di server tersebut.

OK, siapkan file tujuan transfer terlebih dahulu.

```
fDestination, err := sftpClient.Create("/data/nginx/files/test-file.txt")
if err != nil {
    log.Fatal("Failed to create destination file. " + err.Error())
}
```

Lalu baca file di lokal, gunakan `io.Copy` untuk mengcopy isi file.

```
fSource, err := os.Open("/Users/novalagung/Desktop/test-file.txt")
if err != nil {
    log.Fatal("Failed to read source file. " + err.Error())
}

_, err = io.Copy(fDestination, fSource)
if err != nil {
    log.Fatal("Failed copy source file into destination file. " + err.Error())
}

log.Println("File copied.")
```

Jalankan aplikasi untuk melihat hasilnya.

- 
- [sftp](#), by pkg team, BSD-2-Clause License

## CB.28. Web Socket: Chatting App

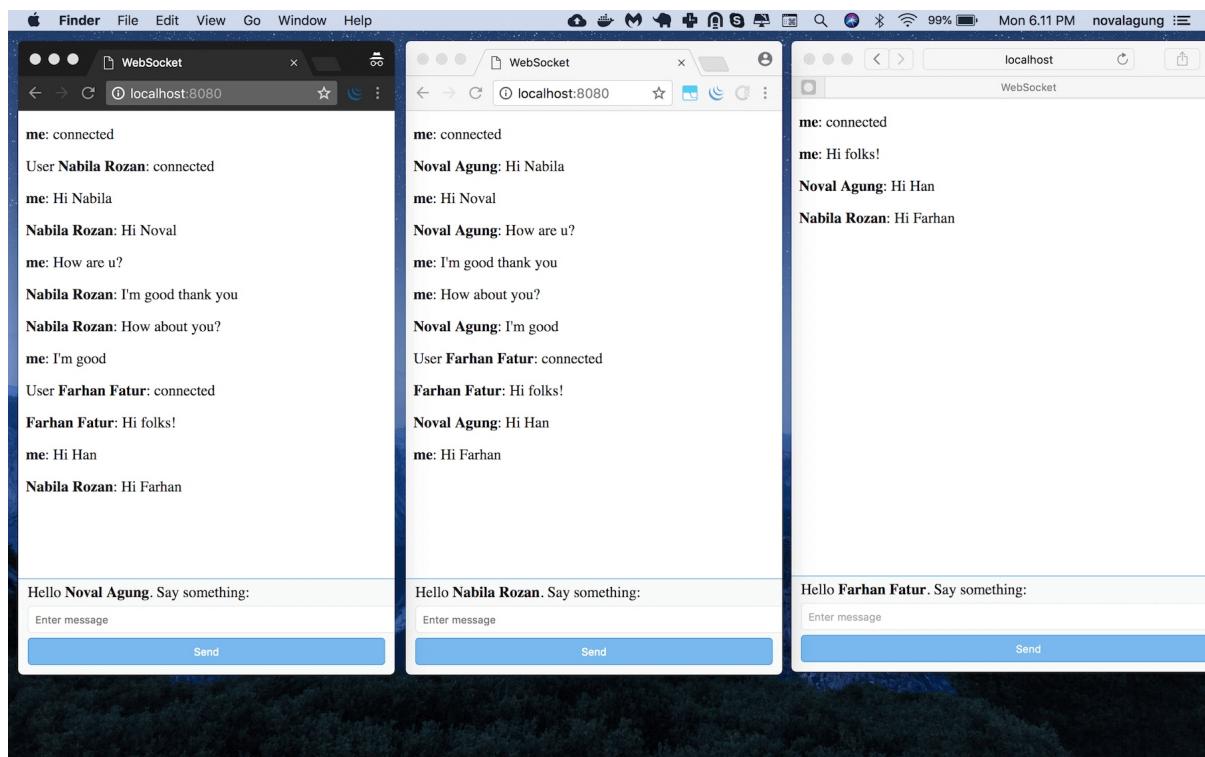
Pada bab ini kita akan belajar penerapan web socket di golang, untuk membuat sebuah aplikasi chatting. Web socket server dibuat menggunakan library [Gorilla Web Socket](#), dan di sisi front end kita menggunakan native API milik javascript yaitu [WebSocket](#) untuk melakukan komunikasi dengan socket server.

Jelasnya kapabilitas web socket bisa dicapai dengan cukup menggunakan default package yang disediakan golang. Namun pada bab ini pembelajaran dilakukan menggunakan 4rd party library.

Seperti biasanya proses belajar dilakukan sambil praktik. Kita buat aplikasi chatting minimalis, dengan kode sedikit mungkin agar mudah dipahami, development dilakukan *from scratch*.

Nantinya saat testing akan ada banyak user terhubung dengan socket server, dalam satu room. Setiap pesan yang ditulis oleh salah seorang user, bisa dibaca oleh semua user lainnya.

Kurang lebih aplikasi yang kita kembangkan seperti gambar di bawah ini.



### CB.28.1. Back End

Buat folder projek baru, siapkan dua buah file: `main.go` dan `index.html`. Kita akan buat socket server terlebih dahulu. Silakan tulis kode berikut ke dalam `main.go`.

```
package main

import (
    "fmt"
    "github.com/gorilla/websocket"
    "github.com/novalagung/gubrak"
    "io/ioutil"
    "log"
    "net/http"
```

```

    "strings"
)

type M map[string]interface{}`

const MESSAGE_NEW_USER = "New User"
const MESSAGE_CHAT = "Chat"
const MESSAGE_LEAVE = "Leave"

var connections = make([]*WebSocketConnection, 0)

```

Konstanta dengan prefix `MESSAGE_*` adalah representasi dari jenis message yang dikirim dari socket server ke semua client (yang terhubung).

- Konstanta `MESSAGE_NEW_USER`. Ketika ada user baru terhubung ke room, maka sebuah pesan **User XXX: connected** akan muncul. Konstanta ini digunakan oleh socket server dalam mem-broadcast informasi tersebut.
- Konstanta `MESSAGE_CHAT`. Ketika user/client mengirim message/pesan ke socket server, message tersebut kemudian diteruskan ke semua client lainnya oleh socket server. Isi pesan di-broadcast oleh socket server ke semua user yang terhubung menggunakan konstanta ini.
- Konstanta `MESSAGE_LEAVE`. Digunakan oleh socket server untuk menginformasikan semua client lainnya, bahwasanya ada client yang keluar dari room (terputus dengan socket server). Pesan **User XXX: disconnected** dimunculkan.

Selain 3 konstanta di atas, ada variabel `connections`. Variabel ini digunakan untuk menampung semua client yang terhubung ke socket server.

OK, setelah kode di atas ditulis, siapkan tiga buah struct berikut.

- Struct `SocketPayload`, digunakan untuk menampung payload yang dikirim dari front end.

```

type SocketPayload struct {
    Message string
}

```

- Struct `SocketResponse`, digunakan oleh back end (socket server) sewaktu mem-broadcast message ke semua client yang terhubung. Field `Type` akan berisi salah satu dari konstanta dengan prefix `MESSAGE_*`.

```

type SocketResponse struct {
    From     string
    Type    string
    Message string
}

```

- Struct `WebSocketConnection`. Nantinya setiap client yang terhubung, objek koneksi-nya disimpan ke slice `connections` yang tipenya adalah `[]*WebSocketConnection`.

```

type WebSocketConnection struct {
    *websocket.Conn
    Username string
}

```

Selanjutnya buat fungsi `main()`, siapkan satu buah rute, `/`, isinya menampilkan template view `index.html`. Siapkan juga rute `/ws` yang akan menjadi gateway komunikasi socket.

```

func main() {
    http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
        content, err := ioutil.ReadFile("index.html")
        if err != nil {

```

```

        http.Error(w, "Could not open requested file", http.StatusInternalServerError)
        return
    }

    fmt.Fprintf(w, "%s", content)
})

http.HandleFunc("/ws", func(w http.ResponseWriter, r *http.Request) {
    // socket code here
})

fmt.Println("Server starting at :8080")
http.ListenAndServe(":8080", nil)
}

```

Handler `/ws` diisi dengan proses untuk konversi koneksi HTTP ke koneksi web socket. Statement `websocket.Upgrade()` digunakan untuk ini. Pada statement tersebut, parameter ke-4 adalah besar read buffer, sedangkan parameter ke-5 adalah besar write buffer.

```

http.HandleFunc("/ws", func(w http.ResponseWriter, r *http.Request) {
    currentGorillaConn, err := websocket.Upgrade(w, r, w.Header(), 1024, 1024)
    if err != nil {
        http.Error(w, "Could not open websocket connection", http.StatusBadRequest)
    }

    username := r.URL.Query().Get("username")
    currentConn := WebSocketConnection{Conn: currentGorillaConn, Username: username}
    connections = append(connections, &currentConn)

    go handleIO(&currentConn, connections)
})

```

Di sisi client, ketika inisialisasi koneksi web socket, informasi `username` disisipkan sebagai query string. Lalu di back end diambil untuk ditempelkan ke objek koneksi socket (yang kemudian dimasukan ke `connections`).

```
app.ws = new WebSocket("ws://localhost:8080/ws?username=" + name)
```

Objek `currentGorillaConn` yang merupakan objek `current` koneksi web server, kita cast ke tipe `WebSocketConnection`, kemudian ditampung ke `currentConn`. Informasi `username` di atas ditambahkan sebagai identifier dalam objek tersebut.

Slice `connections` menampung semua koneksi web socket, termasuk `currentConn`.

Di akhir handler, fungsi `handleIO()` dipanggil sebagai sebuah goroutine, dalam pemanggilannya objek `currentConn` dan `connections` disisipkan. Tugas fungsi `handleIO()` ini adalah untuk me-manage komunikasi antara client dan server. Proses broadcast message ke semua client yg terhubung dilakukan dalam fungsi ini.

Berikut adalah isi fungsi `handleIO()`.

```

func handleIO(currentConn *WebSocketConnection, connections []*WebSocketConnection) {
    defer func() {
        if r := recover(); r != nil {
            log.Println("ERROR", fmt.Sprintf("%v", r))
        }
    }()
}

broadcastMessage(currentConn, MESSAGE_NEW_USER, "")

for {
    payload := SocketPayload{}
    err := currentConn.ReadJSON(&payload)
    if err != nil {

```

```

        if strings.Contains(err.Error(), "websocket: close") {
            broadcastMessage(currentConn, MESSAGE_LEAVE, "")
            ejectConnection(currentConn)
            return
        }

        log.Println("ERROR", err.Error())
        continue
    }

    broadcastMessage(currentConn, MESSAGE_CHAT, payload.Message)
}
}

```

Ketika koneksi terjalin untuk pertama kalinya, antara socket client dan socket server, fungsi `broadcastMessage()` dipanggil. Semua client yang terhubung (kecuali `currentConn`) dikirim pesan dengan jenis `MESSAGE_NEW_USER`, menginformasikan bahwa ada user baru terhubung ke room.

Selanjutnya, ada perulangan tanpa henti. Statement `currentConn.ReadJSON()` dalam loop adalah blocking. Statement tersebut hanya akan tereksekusi ketika ada payload (berupa message/pesan) dikirim dari socket client. Payload tersebut diterima oleh socket server, kemudian di-broadcast ke semua client yang terhubung (kecuali `currentConn`) dengan jenis message terpilih adalah `MESSAGE_CHAT`. Data message sendiri disisipkan sebagai parameter ke-3 pemanggilan `broadcastMessage()`.

Ketika ada client terputus koneksi dengan socket server, method `.ReadJSON()` otomatis terpanggil, namun tidak melakukan apa-apa dan **pasti** mengembalikan error. Berikut adalah error message-nya.

```
websocket: close 1001 (going away)
```

Error diatas adalah indikator bahwa *current* client terputus koneksi dengan socket server. Ketika hal ini terjadi, maka akan ada message yang di-broadcast ke semua client yang terhubung (kecuali `currentConn`) dengan jenis message adalah `MESSAGE_LEAVE`, untuk menginformasikan bahwa ada user (yaitu `currentConn`) yang leave room. Tak lupa, objek `currentConn` dikeluarkan dari slice `connections` lewat fungsi `ejectConnection()`.

Berikut adalah deklarasi fungsi `ejectConnection()` dan `broadcastMessage()`.

- Fungsi `ejectConnection()`:

```

func ejectConnection(currentConn *WebSocketConnection) {
    filtered, _ := gubrak.Reject(connections, func(each *WebSocketConnection) bool {
        return each == currentConn
    })
    connections = filtered.([]*WebSocketConnection)
}

```

- Fungsi `broadcastMessage()`:

```

func broadcastMessage(currentConn *WebSocketConnection, kind, message string) {
    for _, eachConn := range connections {
        if eachConn == currentConn {
            continue
        }

        eachConn.WriteJSON(SocketResponse{
            From:   currentConn.Username,
            Type:   kind,
            Message: message,
        })
    }
}

```

Method `.writeJSON()` milik `websocket.conn` digunakan untuk mengirim data dari socket server ke socket client (yang direpresentasikan oleh `eachConn`). Dalam fungsi `broadcastMessage()` di atas, semua client yang terhubung dikirim data (sesuai parameter), terkecuali untuk current client.

Bagian back end sudah cukup. Sekarang lanjut ke layer front end.

## CB.28.2. Front End

Siapkan terlebih dahulu basis layout front end. Ada dua section penting yg harus disiapkan.

1. Sebuah div dengan ukuran besar, nantinya diisi dengan message yang dikirim oleh current client dan client lainnya.
2. Sebuah form dengan isi satu inputan text dan satu button. Nantinya user menulis pesan yang ingin di-broadcast ke inputan text, lalu klik button untuk submit message tersebut.

Kurang lebih kode-nya seperti berikut.

```
<!DOCTYPE html>
<html>
<head>
    <title>WebSocket</title>

    <style type="text/css">
        // styles here
    </style>
</head>
<body>
    <div class="container"></div>

    <div class="form">
        <form onsubmit="app.doSendMessage(); return false;">
            <div class="placeholder">
                <label>Hello <b class="username"></b>. Say something:</label>
            </div>
            <input class="input-message" type="text" placeholder="Enter message">
            <button type="submit">Send</button>
        </form>
    </div>

    <script type="text/javascript">
        // js script here
    </script>
</body>
</html>
```

Tambahkan beberapa stylesheet agar terlihat cantik.

```
.form {
    position: fixed;
    left: 0;
    bottom: 0;
    right: 0;
    background-color: #f9f9f9;
    border-top: 1px solid #78b8ef;
    padding: 5px 10px;
}

.form .placeholder, .form .input-message, .form button {
    display: block;
    margin-bottom: 5px;
}

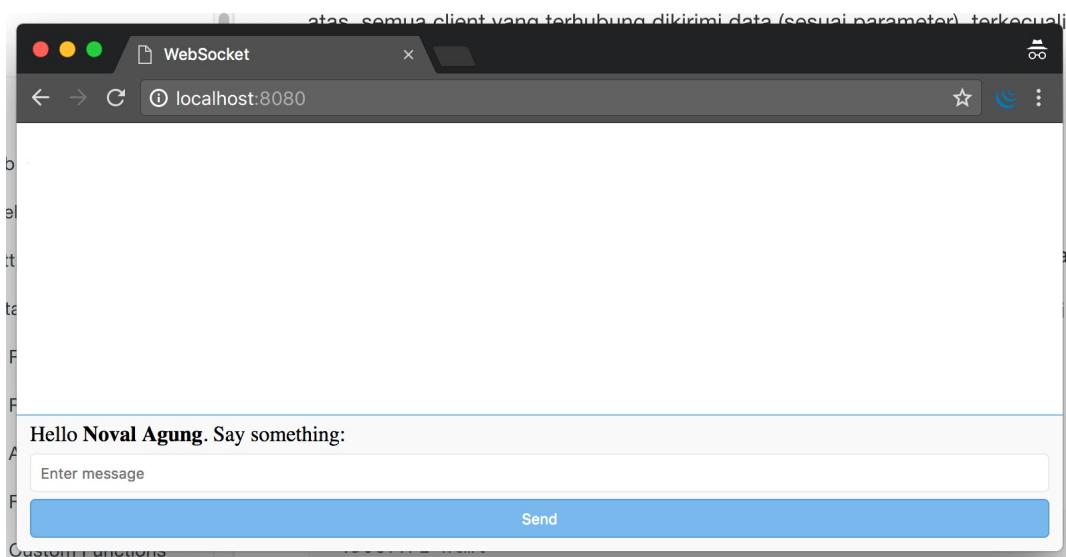
.form .input-message {
    padding: 7px;
```

```

border: 1px solid #ecepbe;
border-radius: 4px;
width: -webkit-fill-available;
}
.form button {
width: 100%;
color: white;
padding: 7px 10px;
border-radius: 4px;
background-color: #78b8ef;
border: 1px solid #5a9ed8;
}
.container { margin-bottom: 50px; }
.container p { display: block; }

```

Tampilan sekilas aplikasi bisa dilihat pada gambar di bawah ini.



OK, sekarang saatnya masuk ke bagian yang paling disukai anak jaman now (?), yaitu javascript. Siapkan beberapa property, satu untuk menampung objek client socket server, dan satu lagi menampung element container (element inilah yang nantinya akan diisi message yang di-broadcast oleh server).

```

var app = {}
app.ws = undefined
app.container = undefined

app.init = function () {
if (!(window.WebSocket)) {
alert('Your browser does not support WebSocket')
return
}

var name = prompt('Enter your name please:') || "No name"
document.querySelector('.username').innerText = name

app.container = document.querySelector('.container')

app.ws = new WebSocket("ws://localhost:8080/ws?username=" + name)

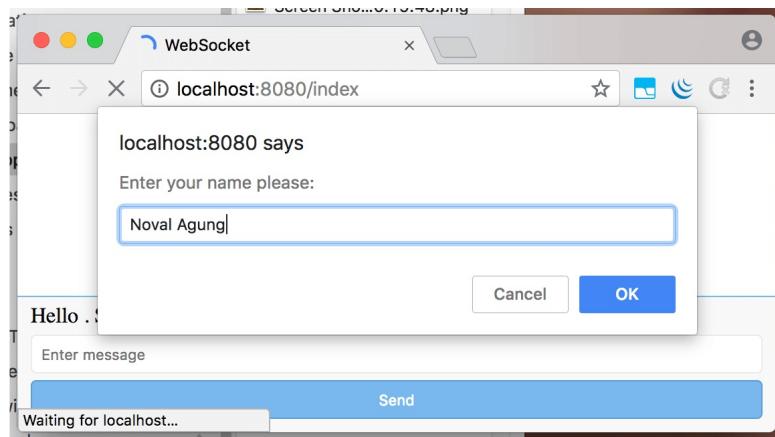
// ...
}

window.onload = app.init

```

Fungsi `app.init()` dipanggil pada event `window.onload`.

Di saat pertama kali page load, muncul prompt yang meminta inputan nama user. Nantinya user yang diinput dijadikan sebagai *current username* pada aplikasi chatting ini.



Property `app.ws` digunakan untuk menampung objek client web socket. Dari objek tersebut, buat 3 buah event listener. Tulis deklarasi event-nya dalam `app.init`.

- Event `onopen`. Event ini dieksekusi ketika *current socket client* berhasil terhubung dengan socket server.

```
app.ws.onopen = function() {
    var message = '<b>me</b>: connected'
    app.print(message)
}
```

- Event `onmessage`. Event ini terpanggil ketika socket server mengirim data dan diterima oleh masing-masing socket client. Di dalam event ini, message yang di-broadcast oleh socket server ditampilkan sesuai jenis message-nya, apakah `New User`, `Leave`, atau `Chat`.

```
app.ws.onmessage = function (event) {
    var res = JSON.parse(event.data)

    var messsage = ''
    if (res.Type === 'New User') {
        message = 'User <b>' + res.From + '</b>: connected'
    } else if (res.Type === 'Leave') {
        message = 'User <b>' + res.From + '</b>: disconnected'
    } else {
        message = '<b>' + res.From + '</b>: ' + res.Message
    }

    app.print(message)
}
```

- Event `onclose`. Seperti yang sudah disinggung di atas, ketika koneksi *current socket* terputus dengan server, event ini terpanggil secara otomatis.

```
app.ws.onclose = function () {
    var message = '<b>me</b>: disconnected'
    app.print(message)
}
```

Kemudian tambahkan fungsi `app.print()`, fungsi ini digunakan untuk mencetak pesan ke `.container`.

```
app.print = function (message) {
    var el = document.createElement("p")
    el.innerHTML = message
}
```

```
    app.container.append(el)
}
```

Dan fungsi `app.doSendMessage()`, fungsi ini digunakan untuk mengirim payload message (inputan user) ke socket server.

```
app.doSendMessage = function () {
  var messageRaw = document.querySelector('.input-message').value
  app.ws.send(JSON.stringify({
    Message: messageRaw
  }));

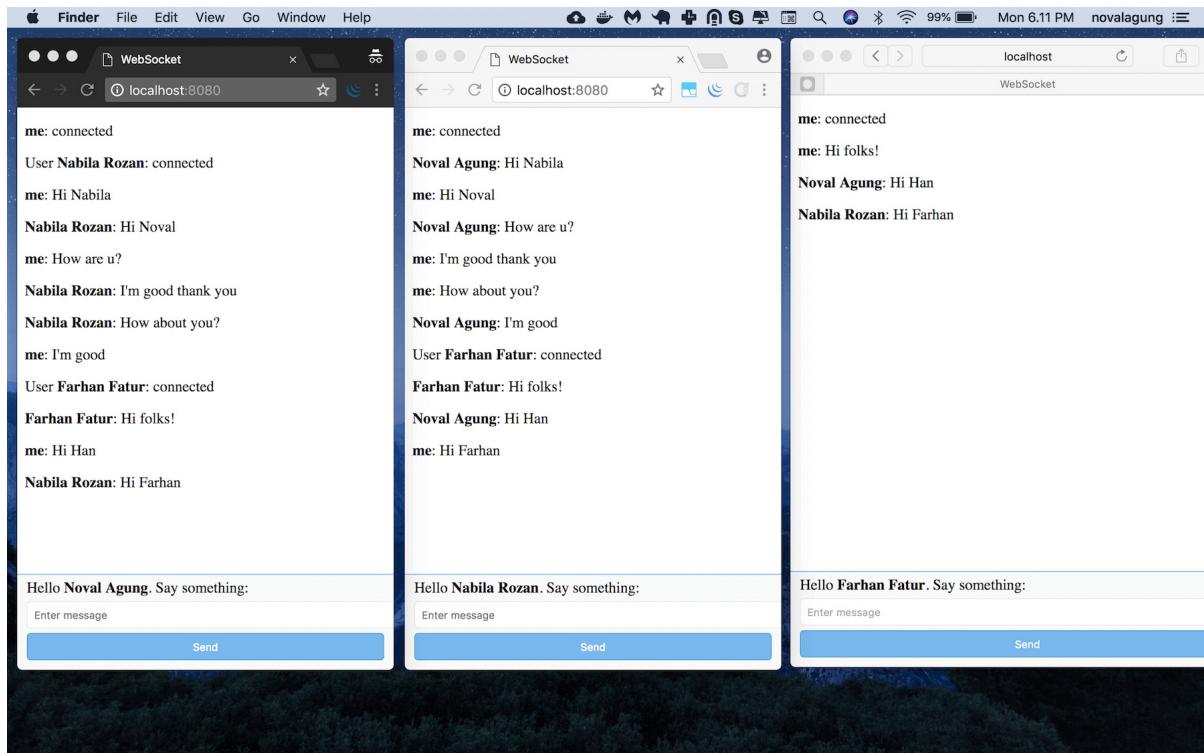
  var message = '<b>me</b>: ' + messageRaw
  app.print(message)

  document.querySelector('.input-message').value = ''
}
```

OK, aplikasi sudah siap, mari lanjut ke bagian testing.

## CB.28.3. Testing

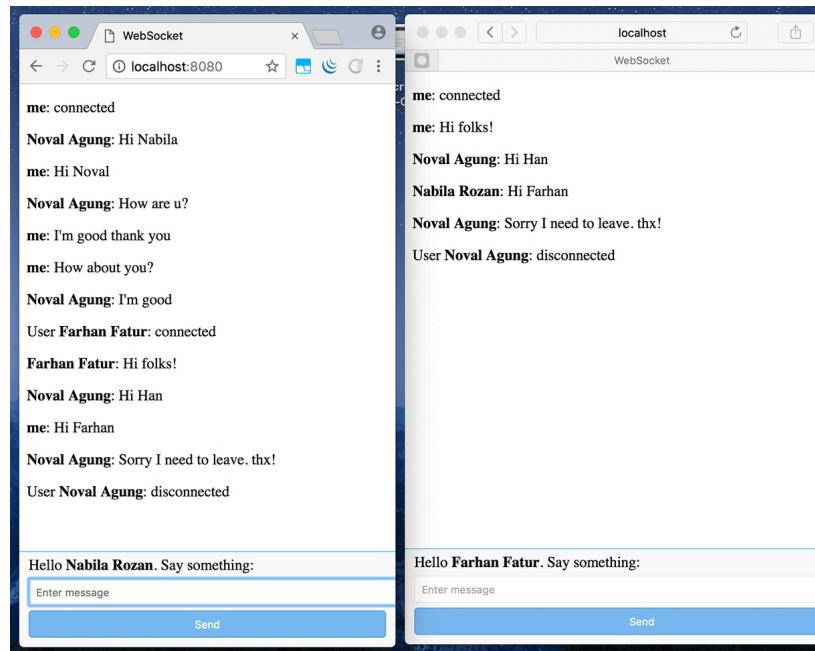
Buka beberapa tab, gunakan username apa saja di masing-masing tab. Coba berinteraksi satu sama lain.



Bisa dilihat, ketika ada user baru, semua client yang sudah terhubung mendapat pesan **User XXX: connected**.

Pesan yang ditulis oleh satu client bisa dilihat oleh client lainnya.

Ketika salah satu user leave, pesan **User XXX: disconnected** akan di-broadcast ke semua user lainnya. Pada gambar di bawah ini dicontohkan user **Noval Agung** leave.



- [Gorilla Web Socket](#), by Gary Burd, BSD-2-Clause License
- [Gubrak](#), by Noval Agung, MIT License

## C.29. Protobuf

Pada bab ini kita akan belajar tentang penggunaan protobuf (Protocol Buffers) di golang.

Perlu diketahui, topik gRPC tidak kita pelajari pada bab ini, melainkan pada bab selanjutnya.

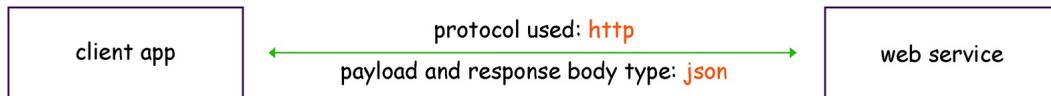
### C.29.1. Definisi

Protocol Buffers adalah metode untuk serialisasi data terstruktur, yang dibuat oleh Google. Protobuf cocok digunakan pada aplikasi yang berkomunikasi dengan aplikasi lain. Protobuf bisa dipakai di banyak platform, contoh: komunikasi antara aplikasi mobile iOS dan Golang Web Service, bisa menggunakan protobuf.

Protobuf hanya bertugas di bagian serialisasi data saja, untuk komunikasi antar service atau antar aplikasi sendiri menggunakan gRPC.

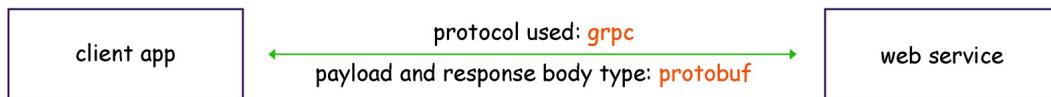
gRPC adalah sebuah remote procedure call atau RPC yang dibuat oleh google. gRPC menggunakan HTTP/2 untuk komunikasinya, dan Protocol Buffers di bagian antarmuka-nya.

Mungkin sampai sini masih terasa abstrak, membingungkan, dan muncul banyak pertanyaan mengenai apa dan untuk apa protobuf ini. Agar lebih mudah untuk dipahami, bayangkan sebuah aplikasi client yang mengkonsumsi data dari (RESTful) web service, dengan data dikirimkan dalam bentuk JSON.



Di analogi sederhana di gambar, dijelaskan bahwa HTTP digunakan sebagai transport antara client dan server, dan JSON digunakan sebagai tipe data payload request dan response body type.

Arsitektur di atas (yang menggunakan http dan json) jika dikonversi ke bentuk gRPC dan protobuf, maka kurang lebih jadinya seperti gambar di bawah ini.



Cukup bisa dipahami bukan?

Sederhananya protobuf itu mirip seperti JSON atau XML, tapi lebih terstruktur. Perbedaannya adalah pada protobuf skema model harus didefinisikan di awal (*schema on write*).

Skema tersebut didefinisikan dalam file berekstensi `.proto`. Dari file tersebut di generate-lah file model sesuai bahasa yang dipilih. Misalkan bahasa yang digunakan adalah java, maka nantinya terbentuk pojo; misal bahasa adalah php nantinya class di-generate; jika bahasa golang maka struct di-generate, dan seterusnya.

gRPC dan protobuf adalah salah satu pilihan terbaik untuk diterapkan pada aplikasi yang mengadopsi konsep microservices.

## C.29.2. Instalasi

Schema yang ditulis dalam `.proto` di-compile ke bentuk file sesuai bahasa yang dipilih. Dari sini jelasnya sebuah compiler dibutuhkan, maka dari itu protobuf harus install terlebih dahulu di lokal masing-masing. Compiler tersebut bernama `protoc` atau proto compiler.

Silakan merujuk ke <http://google.github.io/proto-lens/installing-protoc.html> untuk mengetahui cara instalasi `protoc` sesuai sistem operasi yang dibutuhkan.

Selain `protoc`, masih ada satu lagi yang perlu di install, yaitu protobuf runtime untuk golang (karena di sini kita menggunakan bahasa golang). Cara instalasinya cukup mudah:

```
$ go get -u github.com/golang/protobuf/proto
$ go get -u github.com/golang/protobuf/protoc-gen-go
```

Protobuf runtime tersedia untuk banyak bahasa selain golang, lengkapnya silakan cek <https://github.com/protocolbuffers/protobuf>.

## C.29.3. Pembuatan File `.Proto`

Siapkan satu buah folder dengan skema seperti berikut.

```
novalagung:src $ tree .
.
└── yourproject
    ├── main.go
    └── model
        ├── garage.proto
        └── user.proto
```

Folder `yourproject/model` berisikan file-file `.proto` (dua buah file proto didefinisikan). Dari kedua file di atas akan di-generate file `model.go` menggunakan command `protoc`. Nantinya generated file tersebut dipakai dalam `main.go`.

- **File `user.proto`**

OK, mari kita masuk ke bagian tulis-menulis kode. Buka file `user.proto`, tulis kode berikut.

```
syntax = "proto3";

package model;

enum UserGender {
    UNDEFINED = 0;
    MALE = 1;
    FEMALE = 2;
}
```

Bahasa yang digunakan dalam file proto sangat minimalis, dan cukup mudah untuk dipahami.

Statement `syntax = "proto3";`, artinya adalah versi proto yang digunakan adalah `proto3`. Ada juga versi `proto2`, namun kita tidak menggunakananya.

Statement `package model;`, digunakan untuk menge-set nama package dari file yang nantinya di-generate. File `user.proto` akan di-compile, menghasilkan file `user.pb.go`. File golang tersebut package-nya adalah sesuai dengan yang sudah didefinisikan di statement, pada contoh ini `model`.

Statement `enum UserGender` digunakan untuk pendefinisian enum. Tulis nama-nama enum beserta value di dalam kurung kurawal. Keyword `UserGender` sendiri nantinya menjadi tipe enum. Value enum di protobuf hanya bisa menggunakan tipe data numerik int.

Setelah proses kompilasi ke bentuk golang, kode di atas kurang lebihnya akan menjadi seperti berikut.

```
package model

type UserGender int32

const (
    UserGender_UNDEFINED UserGender = 0
    UserGender_MALE     UserGender = 1
    UserGender_FEMALE   UserGender = 2
)
```

Selanjutnya tambahkan statement pendefinisian message berikut dalam file `user.proto`.

```
message User {
    string id = 1;
    string name = 2;
    string password = 3;
    UserGender gender = 4;
}

message UserList {
    repeated User list = 1;
}
```

Untuk mengurangi moskimunikasi, penulis gunakan istilah `model` untuk `message` pada kode di atas.

Model didefinisikan menggunakan keyword `message` diikuti dengan nama model-nya. Di dalam kurung kurawal, ditulis property-property model dengan skema penulisan `<tipe data> <nama property> = <numbered field>`.

Bisa dilihat bahwa di tiap field terdapat *unique number*. Informasi ini berguna untuk versioning model protobuf.

Tiap field harus memiliki angka yang unik satu sama lain dalam satu `message`.

Di dalam `User`, dideklarasikan property `id`, `name`, dan `password` yang bertipe `string`; dan juga property `gender` yang bertipe `UserGender`.

Selain `User`, model `UserList` juga didefinisikan. Isinya hanya satu property yaitu `list` yang tipe-nya adalah `User`. Keyword `repeated` pada property digunakan untuk pendefinisian tipe array atau slice. Statement `repeated User` adalah ekuivalen dengan `[]*User` (tipe element slice pasti pointer).

Kode protobuf di atas menghasilkan kode golang berikut.

```
type User struct {
    Id      string
    Name   string
    Password string
    Gender  UserGender
}

type UserList struct {
    List []User
}
```

### • File garage.proto

Sekarang beralih ke file ke-dua, `garage.proto`. Silakan tulis kode berikut.

```
syntax = "proto3";

package model;

message GarageCoordinate {
    float latitude = 1;
    float longitude = 2;
}

message Garage {
    string id = 1;
    string name = 2;
    GarageCoordinate coordinate = 3;
}
```

Bisa dilihat, property `coordinate` pada model `Garage` tipe-nya adalah model juga, yaitu `GarageCoordinate`.

Di atas, tipe data `float` digunakan pada pendefinisian property `latitude` dan `longitude`. Silakan merujuk ke link berikut untuk mengetahui tipe-tipe primitif apa saja yang bisa digunakan sebagai tipe data property model <https://developers.google.com/protocol-buffers/docs/proto3#scalar>.

Buat dua buah model lainnya berikut ini.

```
message GarageList {
    repeated Garage list = 1;
}

message GarageListByUser {
    map<string, GarageList> list = 1;
}
```

Selain array/slice, tipe `map` juga bisa digunakan pada protobuf. Gunakan keyword `map` untuk mendefinisikan tipe map. Penulisannya disertai dengan penulisan tipe data key dan tipe data value map tersebut.

Penulisan tipe data map mirip seperti penulisan `HashMap` pada java yang disisipkan juga tipe generics-nya.

Kembali ke topik, dua message di atas akan menghasilkan kode golang berikut.

```
type GarageList struct {
    List []*Garage
}

type GarageListByUser struct {
    List map[string]*GarageList
}
```

## C.29.4. Kompilasi File .Proto

File `.proto` sudah siap, sekarang saatnya untuk meng-compile file-file tersebut agar menjadi `.go`. Kompilasi dilakukan lewat command `protoc`. Agar output berupa file golang, maka gunakan flag `--go_out`. Lebih jelasnya silakan ikut command berikut.

```
novalagung:model $ protoc --go_out=. *.proto
novalagung:model $ tree .
```

```

|--- garage.pb.go
|--- garage.proto
|--- user.pb.go
|--- user.proto

0 directories, 4 files

```

Dua file baru dengan ekstensi `.pb.go` muncul.

## C.29.5. Praktek

Sekarang kita akan belajar tentang pengoperasian file proto yang sudah di-generate. Buka file `main.go`, tulis kode berikut.

```

package main

import (
    "bytes"
    "fmt"
    "os"

    // sesuaikan dengan struktur folder projek masing2
    "dasarpemrogramangolang/chapter-B.29/model"
)

func main() {
    // more code here ...
}

```

Package `model` yang isinya generated proto file, di-import. Dari package tersebut, kita bisa mengakses generated struct seperti `model.User`, `model.GarageList`, dan lainnya. Maka coba buat beberapa objek untuk kesemua generated struct.

- Objek struct `model.User` :

```

var user1 = &model.User{
    Id:      "u001",
    Name:   "Sylvana Windrunner",
    Password: "f0r Th3 H0rd3",
    Gender:  model.UserGender_FEMALE,
}

```

Untuk tipe enum pengaksesannya seperti di atas, penulisannya `model.UserGender_*`. Cukup ubah `*` dengan value yang diinginkan, `UNDEFINED`, `MALE`, atau `FEMALE`.

- Objek struct `model.UserList` :

```

var userList = &model.UserList{
    List: []*model.User{
        user1,
    },
}

```

Disarankan untuk selalu menampung objek cetakan struct protobuf dalam bentuk pointer, karena dengan itu objek akan memenuhi kriteria interface `proto.Message`, yang nantinya akan sangat membantu proses coding.

- Objek struct `model.Garage` :

```
var garage1 = &model.Garage{
    Id: "g001",
    Name: "Kalim dor",
    Coordinate: &model.GarageCoordinate{
        Latitude: 23.2212847,
        Longitude: 53.22033123,
    },
}
```

- Objek struct `model.GarageList` :

```
var garageList = &model.GarageList{
    List: []*model.Garage{
        garage1,
    },
}
```

- Objek struct `model.GarageListByUser` :

```
var garageListByUser = &model.GarageListByUser{
    List: map[string]*model.GarageList{
        user1.Id: garageList,
    },
}
```

## • Print proto object

Print salah satu objek yang sudah dibuat di atas.

```
// ===== original
fmt.Printf("# === Original\n      %#v \n", user1)

// ===== as string
fmt.Printf("# === As String\n      %v \n", user1.String())
```

Jalankan aplikasi untuk melihat hasilnya.

```
[novalagung:chapter-B.29 $ go run main.go
# === Original
&model.User{Id:"u001", Name:"Sylvana Windrunner", Password:"f0r Th3 H0rD3", Gender:2,
XXX_NoUnkeyedLiteral:struct {}{}, XXX_unrecognized:[]uint8(nil), XXX_sizecache:0}
# === As String
id:"u001" name:"Sylvana Windrunner" password:"f0r Th3 H0rD3" gender:FEMALE
novalagung:chapter-B.29 $ ]
```

Pada statement print pertama, objek ditampilkan apa adanya. Generated struct memiliki beberapa property lain selain yang sudah didefinisikan pada proto message, seperti `XXX_unrecognized` dan beberapa lainnya. Property tersebut dibutuhkan oleh protobuf, tapi tidak kita butuhkan, jadi biarkan saja.

Di statement print kedua, method `.String()` diakses, menampilkan semua property yang didefinisikan dalam proto message (property `xxx_` tidak dimunculkan).

## • Konversi objek proto ke json string

Tambahkan kode berikut:

```
// ===== as json string
var buf bytes.Buffer
err1 := (&jsonpb.Marshaler{}).Marshal(&buf, garageList)
if err1 != nil {
    fmt.Println(err1.Error())
    os.Exit(0)
}
jsonString := buf.String()
fmt.Printf("# === As JSON String\n      %v \n", jsonString)
```

Di atas kita membuat banyak objek lewat generated struct. Objek tersebut bisa dikonversi ke bentuk JSON string, caranya dengan memanfaatkan package [github.com/golang/protobuf/jsonpb](https://github.com/golang/protobuf/jsonpb). Lakukan `go get` pada package jika belum punya, dan jangan lupa untuk meng-importnya pada file yang sedang dibuat.

Kembali ke pembahasan, buat objek pointer baru dari struct `jsonpb.Marshaler`, lalu akses method `.Marshal()`. Sisipkan objek bertipe `io.Writer` penampung hasil konversi sebagai parameter pertama (pada contoh di atas kita gunakan `bytes.Buffer`). Lalu sisipkan juga objek proto yang ingin dikonversi sebagai parameter kedua.

Jalankan aplikasi, cek hasilnya.

```
# ===== Original
  &model.User{Id:"u001", Name:"Sylvana Windrunner", Password:"f0r Th3 H0rD3", Gender:2, XXX_NoUnkeyedL
iteral:struct {}{}, XXX_unrecognized:[]uint8(nil), XXX_sizecache:0}
# ===== As String
  id:"u001" name:"Sylvana Windrunner" password:"f0r Th3 H0rD3" gender:FEMALE
# ===== As JSON String
  {"list":[{"id":"g001","name":"Kalimdor","coordinate":{"latitude":23.221285,"longitude":53.220333}]}}
novalagung:chapter-B.29 $
```

Selain method `.Marshal()`, konversi ke json string bisa dilakukan lewat method `.MarshalToString()`.

## • Konversi json string ke objek proto

Proses unmarshal dari json string ke objek proto, bisa dilakukan lewat dua cara:

- Membuat objek pointer dari struct `jsonpb.Unmarshaler`, lalu mengakses method `.Unmarshal()`. Parameter pertama diisi dengan objek `io.Reader` yang isinya json string, dan parameter kedua adalah receiver.

```
buf2 := strings.NewReader(jsonString)
protoObject := new(model.GarageList)

err2 := (&jsonpb.Unmarshaler{}).Unmarshal(buf2, protoObject)
if err2 != nil {
    fmt.Println(err2.Error())
    os.Exit(0)
}

fmt.Printf("# === As String\n      %v \n", protoObject.String())
```

- Menggunakan `jsonpb.UnmarshalString`, dengan parameter pertama disisipi data json string.

```
protoObject := new(model.GarageList)

err2 := jsonpb.UnmarshalString(jsonString, protoObject)
if err2 != nil {
    fmt.Println(err2.Error())
    os.Exit(0)
}

fmt.Printf("# === As String\n      %v \n", protoObject.String())
```

Silakan pilih cara yang cocok sesuai kebutuhan. Lalu jalankan aplikasi dan lihat hasilnya.

```
[novalagung:chapter-B.29 $ go run main.go
# ===== Original
<     &model.User{Id:"u001", Name:"Sylvana Windrunner", Password:"f0r Th3 H0rD3", Gender:2, XXX_NoUnkeyedL
teral:struct {}, XXX_unrecognized:[]uint8(nil), XXX_sizecache:0}
# ===== As String
    id:"u001" name:"Sylvana Windrunner" password:"f0r Th3 H0rD3" gender:FEMALE
# ===== As JSON String
    {"list":[{"id":"g001","name":"Kalimdor","coordinate":{"latitude":23.221285,"longitude":53.220333}]}]
# ===== As String
    list:<id:"g001" name:"Kalimdor" coordinate:<latitude:23.221285 longitude:53.220333 > >
novalagung:chapter-B.29 $ ]
```

## C.29.6. gRPC + Protobuf

Pada bab selanjutnya kita akan belajar tentang penerapan gRPC dan protobuf.

---

- [Protobuf](#), by Google, BSD-3-Clause License

## C.30. gRPC + Protobuf

Pada bab ini kita akan belajar tentang penerapan gRPC dan protobuf pada bahasa goLang.

Kita akan buat satu buah folder projek besar, didalamnya terdapat 3 buah aplikasi. Dua diantaranya merupakan aplikasi server, rpc server. Dan yang satu adalah aplikasi client. Aplikasi client akan berkomunikasi dengan kedua aplikasi server.

Bisa dibilang ini adalah contoh super sederhana (dan asal-asalan) tentang penerapan [microservices architecture](#).

**BEWARE:** Tutorial ini sangat panjang! Dan jangan ber-ekspektasi terlalu tinggi, karena target pembaca adalah orang yang baru belajar goLang, gRPC, dan protobuf.

### C.30.1. Definisi

gRPC adalah RPC framework buatan google. gRPC menggunakan RPC untuk transport dan protobuf di bagian antarmuka-nya.

Remote Procedure Call (RPC) adalah sebuah metode yang memungkinkan kita untuk mengakses sebuah prosedur yang berada di komputer lain. Untuk dapat melakukan ini sebuah server harus menyediakan layanan remote procedure.

### C.30.2. Struktur Aplikasi

Siapkan satu project baru dengan struktur sebagai berikut.

```
novalagung:chapter-b.30 $ tree .
.
├── common
│   ├── config
│   │   └── config.go
│   └── model
│       ├── garage.proto
│       ├── user.proto
│       └── util.proto
|
└── services
    ├── service-garage
    │   └── main.go
    └── service-user
        └── main.go

7 directories, 7 files
```

Projek kali ini cukup kompleks, dibawah ini merupakan penjelasan per bagian dari struktur projek di atas.

- **Folder common**

Folder `common`, berisikan 2 buah sub folder, `config` dan `model`.

- Folder `config` berisikan informasi shared atau global, yang digunakan aplikasi client maupun server.
- Folder `model` berisikan file `.proto`. Silakan salin file `garage.proto` dan `user.proto` pada bab sebelumnya ke

folder tersebut.

#### • Folder client

Isinya adalah satu buah file main, yang nantinya di jalankan sebagai aplikasi client. Aplikasi client ini akan berkomunikasi dengan 2 buah aplikasi server.

#### • Folder services

Satu buah file proto untuk satu aplikasi rpc server (service). Karena ada dua file proto, berarti jelasnya ada dua aplikasi rpc server, `service-user` dan `service-garage`. Folder `services` ini menampung kedua aplikasi service tersebut.

File `garage.proto` dan `user.proto` tidak dijadikan satu dalam respektif folder `service-garage` dan `service-user`, karena kedua file ini juga digunakan pada aplikasi client, itulah kenapa file ini dipisah ke dalam folder `common/model`.

### C.30.3. File Konfigurasi pada folder `common/config`

Siapkan file bernama `config.go` dalam `common/config`. Di dalam file config ini didefinisikan dua buah konstanta yaitu port untuk service user dan garage. Nantinya aplikasi server di start menggunakan port ini.

```
package config

const (
    SERVICE_GARAGE_PORT = ":7000"
    SERVICE_USER_PORT   = ":9000"
)
```

### C.30.4. Proto Model pada folder `common/model`

Setelah kedua file `user.proto` dan `garage.proto` pada bab sebelumnya disalin, kita perlu menambahkan pendefinisian service pada masing-masing file proto.

Keyword `service` digunakan untuk membuat service. Service ini nantinya juga ikut di konversi ke bentuk golang (menjadi interface), lewat command `protoc`. Di aplikasi rpc server, nantinya harus dibuat implementasi dari interface tersebut.

OK, sekarang tambahkan kode berikut ke file proto.

#### • Service Users

Buka file `user.proto`, tambahkan kode berikut di akhir baris.

```
// ...

import "google/protobuf/Empty.proto";

service Users {
    rpc Register(User) returns (google.protobuf.Empty) {}
    rpc List(google.protobuf.Empty) returns (UserList) {}
}
```

Sebuah service bernama `Users` didefinisikan, dengan isi dua buah method.

- `Register()`, menerima parameter bertipe model `User`, dan mengembalikan `google.protobuf.Empty`.
- `List()`, menerima parameter bertipe `google.protobuf.Empty`, dan mengembalikan tipe `UserList`.

Silakan dilihat bagaimana cara untuk membuat service pada kode atas.

Pada method `Register()` sebenarnya kita tidak butuh nilai balik. Tapi karena requirements dari protobuf mewajibkan semua rpc method harus memiliki nilai balik, maka kita gunakan model `Empty` milik google protobuf.

Cara penggunaan model `Empty` adalah dengan meng-import file proto-nya terlebih dahulu,

`google/protobuf/Empty.proto`, lalu menggunakan `google.protobuf.Empty` sebagai model.

Juga, pada method `List()`, sebenarnya argument tidak dibutuhkan, tapi karena protobuf mewajibkan pendefinisian rpc method untuk memiliki satu buah argument dan satu buah return type, maka mau tidak mau harus ada argument.

Setelah di-compile, dua buah interface terbuat dengan skema nama `<interfacename>Server` dan `<interfacename>Client`. Karena nama service adalah `Users`, maka terbuatlah `UsersServer` dan `UsersClient`.

- Interface `UsersServer`.

```
type UsersServer interface {
    Register(context.Context, *User) (*empty.Empty, error)
    List(context.Context, *empty.Empty) (*UserList, error)
}
```

Interface ini nantinya harus diimplementasikan di aplikasi rpc server.

- Interface `UsersClient`.

```
type UsersClient interface {
    Register(ctx context.Context, in *User, opts ...grpc.CallOption) (*empty.Empty, error)
    List(ctx context.Context, in *empty.Empty, opts ...grpc.CallOption) (*UserList, error)
}
```

Interface ini nantinya harus diimplementasikan di aplikasi rpc client.

## • Service Garages

Pada file `garage.proto`, definisikan service `Garages` dengan isi dua buah method, `Add()` dan `List()`.

```
import "google/protobuf/Empty.proto";

service Garages {
    rpc List(string) returns (GarageList) {}
    rpc Add(string, Garage) returns (google.protobuf.Empty) {}
}
```

Perlu diketahui bahwa protobuf mewajibkan pada rpc method untuk tidak menggunakan tipe primitif sebagai tipe argument maupun tipe nilai balik. Tipe `string` pada `rpc List(string)` akan menghasilkan error saat di-compile. Dan juga protobuf mewajibkan method untuk hanya menerima satu buah parameter, maka jelasnya `rpc Add(string, Garage)` juga invalid.

Lalu bagaimana solusinya? Buat model baru lagi, property nya sesuaikan dengan parameter yang dibutuhkan di masing-masing rpc method.

```
message GarageUserId {
    string user_id = 1;
}

message GarageAndUserId {
```

```

    string user_id = 1;
    Garage garage = 2;
}

service Garages {
    rpc List(GarageUserId) returns (GarageList) {}
    rpc Add(GarageAndUserId) returns (google.protobuf.Empty) {}
}

```

Sama seperti service `users`, service `Garages` juga akan di-compile menjadi interface.

- Interface `GaragesServer`.

```

type GaragesServer interface {
    Add(context.Context, *GarageAndUserId) (*empty.Empty, error)
    List(context.Context, *GarageUserId) (*GarageList, error)
}

```

- Interface `GaragesClient`.

```

type GaragesClient interface {
    Add(ctx context.Context, in *GarageAndUserId, opts ...grpc.CallOption) (*empty.Empty, error)
    List(ctx context.Context, in *GarageUserId, opts ...grpc.CallOption) (*GarageList, error)
}

```

## C.30.5. Kompilasi File `.proto`

Command yang sebelumnya digunakan untuk kompilasi:

```
$ protoc --go_out=. *.proto
```

Command tersebut tidak meng-generate service menjadi interface. Pada argument `--go_out` perlu diubah sedikit agar service dan rpc method ikut di-generate. Gunakan command berikut.

```
$ protoc --go_out=plugins=grpc:. *.proto
```

## C.30.6. Aplikasi Server `service-user`

Buka file `services/service-user/main.go`, import package yang dibutuhkan.

```

package main

import (
    "context"
    "log"
    "net"

    "github.com/golang/protobuf/ptypes/empty"
    "google.golang.org/grpc"
    "dasarpemrogramangolang/chapter-B.30/common/config"
    "dasarpemrogramangolang/chapter-B.30/common/model"
)

```

Lalu buat satu buah objek bernama `localStorage` bertipe `*model.UserList`. Objek ini nantinya menampung data user yang ditambahkan dari aplikasi client (via rpc) lewat method `Register()`. Dan data objek ini juga dikembalikan ke client ketika `List()` diakses.

```
var localStorage *model.UserList

func init() {
    localStorage = new(model.UserList)
    localStorage.List = make([]*model.User, 0)
}
```

Buat struct baru `UsersServer`. Struct ini akan menjadi implementasi dari generated interface `model.UsersServer`. Siapkan method-methodnya sesuai spesifikasi interface.

```
type UsersServer struct{}

func (UsersServer) Register(ctx context.Context, param *model.User) (*empty.Empty, error) {
    localStorage.List = append(localStorage.List, param)

    log.Println("Registering user", param.String())

    return new(empty.Empty), nil
}

func (UsersServer) List(ctx context.Context, void *empty.Empty) (*model.UserList, error) {
    return localStorage, nil
}
```

Buat fungsi `main()`, buat objek grpc server dan objek implementasi `UsersServer`, lalu registrasikan kedua objek tersebut ke model menggunakan statement `model.RegisterUsersServer()`.

```
func main() {
    srv := grpc.NewServer()
    var userSrv UsersServer
    model.RegisterUsersServer(srv, userSrv)

    log.Println("Starting RPC server at", config.SERVICE_USER_PORT)

    // more code here ...
}
```

Pembuatan objek grpc server dilakukan lewat `grpc.NewServer()`. Package [google.golang.org/grpc](https://google.golang.org/grpc) perlu di `go get` dan di-import.

Fungsi `RegisterUsersServer()` otomatis digenerate oleh protoc, karena service `Users` didefinisikan. Contoh lainnya misal service `SomeServiceTest` disiapkan, maka fungsi `RegisterSomeServiceTestServer()` dibuat.

Selanjutnya, siapkan objek listener yang listen ke port `config.SERVICE_USER_PORT`, lalu gunakan listener tersebut sebagai argument method `.Serve()` milik objek rpc server.

```
// ...

l, err := net.Listen("tcp", config.SERVICE_USER_PORT)
if err != nil {
    log.Fatalf("could not listen to %s: %v", config.SERVICE_USER_PORT, err)
}

log.Fatal(srv.Serve(l))
```

## C.30.6. Aplikasi Server service-garage

Buat file `services/service-garage/main.go`, import package yang sama seperti pada `service-user`. Lalu buat objek `localStorage` dari struct `*model.GarageListByUser`.

```
var localStorage *model.GarageListByUser

func init() {
    localStorage = new(model.GarageListByUser)
    localStorage.List = make(map[string]*model.GarageList)
}

type GaragesServer struct{}
```

Tugas `localStorage` kurang lebih sama seperti pada `service-user`, hanya saja pada aplikasi ini data garage disimpan per user dalam map.

Selanjutnya buat implementasi interface `model.GaragesServer`, lalu siapkan method-method-nya.

- Method `Add()`

```
func (GaragesServer) Add(ctx context.Context, param *model.GarageAndUserId) (*empty.Empty, error) {
    userId := param.UserId
    garage := param.Garage

    if _, ok := localStorage.List[userId]; !ok {
        localStorage.List[userId] = new(model.GarageList)
        localStorage.List[userId].List = make([]*model.Garage, 0)
    }
    localStorage.List[userId].List = append(localStorage.List[userId].List, garage)

    log.Println("Adding garage", garage.String(), "for user", userId)

    return new(empty.Empty), nil
}
```

- Method `List()`

```
func (GaragesServer) List(ctx context.Context, param *model.GarageUserId) (*model.GarageList, error) {
    userId := param.UserId

    return localStorage.List[userId], nil
}
```

Start rpc server, gunakan `config.SERVICE_GARAGE_PORT` sebagai port aplikasi.

```
func main() {
    srv := grpc.NewServer()
    var garageSrv GaragesServer
    model.RegisterGaragesServer(srv, garageSrv)

    log.Println("Starting RPC server at", config.SERVICE_GARAGE_PORT)

    l, err := net.Listen("tcp", config.SERVICE_GARAGE_PORT)
    if err != nil {
        log.Fatalf("could not listen to %s: %v", config.SERVICE_GARAGE_PORT, err)
    }

    log.Fatal(srv.Serve(l))
}
```

## C.30.7. Aplikasi Client & Testing

Buat file `client/main.go`, import package yang sama seperti pada `service-user` maupun `service-garage`. Lalu siapkan dua buah method yang mengembalikan rpc client yang terhubung ke dua service yang sudah kita buat.

- RPC client garage:

```
func serviceGarage() model.GaragesClient {
    port := config.SERVICE_GARAGE_PORT
    conn, err := grpc.Dial(port, grpc.WithInsecure())
    if err != nil {
        log.Fatal("could not connect to", port, err)
    }

    return model.NewGaragesClient(conn)
}
```

- RPC client user:

```
func serviceUser() model.UsersClient {
    port := config.SERVICE_USER_PORT
    conn, err := grpc.Dial(port, grpc.WithInsecure())
    if err != nil {
        log.Fatal("could not connect to", port, err)
    }

    return model.NewUsersClient(conn)
}
```

Buat fungsi `main()`, isi dengan pembuatan object dari generated-struct yang ada di package `model`.

```
func main() {
    user1 := model.User{
        Id:      "n001",
        Name:    "Noval Agung",
        Password: "kw8d hl12/3m,a",
        Gender:  model.UserGender(model.UserGender_value["MALE"]),
    }

    garage1 := model.Garage{
        Id:      "q001",
        Name:    "Quel'thalas",
        Coordinate: &model.GarageCoordinate{
            Latitude: 45.123123123,
            Longitude: 54.1231313123,
        },
    }

    // ...
}
```

### • Test rpc client user

Selanjutnya akses fungsi `serviceUser()` untuk memperoleh objek rpc client user. Dari situ eksekusi method `.Register()`.

```
user := serviceUser()

fmt.Println("\n", "===== user test")

// register user1
```

```
user.Register(context.Background(), &user1)

// register user2
user.Register(context.Background(), &user2)
```

Jalankan aplikasi `service-user`, `service-garage`, dan `client`, lalu lihat hasilnya.

```
[novalagung:service-garage $ go run main.go
2018/09/14 14:19:28 Starting RPC server at :7000
[novalagung:service-user $ go run main.go
2018/09/14 14:19:32 Starting RPC server at :9000
2018/09/14 14:19:44 Registering user id:"n001" name:"Noval Agung" password:"kw8d hl12/3m,a" gender:MALE
2018/09/14 14:19:44 Registering user id:"n002" name:"Nabila Rozan" password:"Password Tralala" gender:FEMALE
[novalagung:client $ go run main.go
=====> user test
novalagung:client $ ]
```

Bisa dilihat, pemanggilan method `Add()` pada aplikasi rpc server `service-user` sukses.

Sekarang coba panggil method `.List()`. Jalankan ulang aplikasi `client` untuk melihat hasilnya. O iya, aplikasi `service-user` juga di-restart, agar datanya tidak menumpuk.

```
// show all registered users
res1, err := user.List(context.Background(), new(empty.Empty))
if err != nil {
    log.Fatal(err.Error())
}
res1String, _ := json.Marshal(res1.List)
log.Println(string(res1String))
```

Bisa dilihat pada gambar berikut, pemanggilan method `.List()` juga sukses. Dua buah data yang sebelumnya didaftarkan muncul.

```
[novalagung:service-garage $ go run main.go
2018/09/14 14:19:28 Starting RPC server at :7000
[novalagung:service-user $ go run main.go
2018/09/14 14:25:21 Starting RPC server at :9000
2018/09/14 14:25:24 Registering user id:"n001" name:"Noval Agung" password:"kw8d hl12/3m,a" gender:MALE
2018/09/14 14:25:24 Registering user id:"n002" name:"Nabila Rozan" password:"Password Tralala" gender:FEMALE
[novalagung:client $ go run main.go
=====> user test
2018/09/14 14:25:24 [{"id":"n001","name":"Noval Agung","password":"kw8d hl12/3m,a","gender":1},{"id":"n002","name":"Nabila Rozan","password":"Password Tralala","gender":2}]
novalagung:client $ ]
```

## • Test rpc client garage

Tambahkan beberapa statement untuk memanggil method yang ada di `service-garage`.

- Menambahkan garage untuk user `user1`:

```

garage := serviceGarage()

fmt.Println("\n", "===== garage test A")

// add garage1 to user1
garage.Add(context.Background(), &model.GarageAndUserId{
    UserId: user1.Id,
    Garage: &garage1,
})

// add garage2 to user1
garage.Add(context.Background(), &model.GarageAndUserId{
    UserId: user1.Id,
    Garage: &garage2,
})

```

- Menampilkan list semua garage milik `user1` :

```

// show all garages of user1
res2, err := garage.List(context.Background(), &model.GarageUserId{UserId: user1.Id})
if err != nil {
    log.Fatal(err.Error())
}
res2String, _ := json.Marshal(res2.List)
log.Println(string(res2String))

```

- Menambahkan garage untuk user `user2` :

```

fmt.Println("\n", "===== garage test B")

// add garage3 to user2
garage.Add(context.Background(), &model.GarageAndUserId{
    UserId: user2.Id,
    Garage: &garage3,
})

```

- Menampilkan list semua garage milik `user2` :

```

// show all garages of user2
res3, err := garage.List(context.Background(), &model.GarageUserId{UserId: user2.Id})
if err != nil {
    log.Fatal(err.Error())
}
res3String, _ := json.Marshal(res3.List)
log.Println(string(res3String))

```

Jalankan ulang services dan aplikasi client, lihat hasilnya.

```
[novalagung:service-garage $ go run main.go — NVL — 85x18
2018/09/14 14:19:28 Starting RPC server at :7000
2018/09/14 14:33:40 Adding garage id:"q001" name:"Quel'thalas" coordinate:<latitude:4
5.123123 longitude:54.12313 > for user n001
2018/09/14 14:33:40 Adding garage id:"f001" name:"Frostwing" coordinate:<latitude:4
5.123123 longitude:11.123132 >
[novalagung:client $ go run main.go — NVL — 85x17
2018/09/14 14:33:40 Adding garage id:"q001" name:"Quel'thalas" coordinate:<latitude:4
5.123123 longitude:12.12313 >
=====
[novalagung:service-user $ go run main.go — NVL — 85x17
2018/09/14 14:33:36 Starting
2018/09/14 14:33:40 Register {"id":"n001","name":"Noval Agung","password":"kw8d hl12/3m,a","gender":1}
{"id":"n002","name":"Nabila Rozan","password":"PasswordTralala","gender":2}
2018/09/14 14:33:40 Register {"id":"u001","name":"Undercity","coordinate":{"latitude":22.123123,"longitude":123.12313}}
2018/09/14 14:33:40 Register {"id":"u002","name":"Undercity","coordinate":{"latitude":22.123123,"longitude":123.123132}}
=====
[novalagung:client $ ]
```

OK, jika anda membaca sampai baris ini, berarti anda telah berhasil sabar dalam mempelajari gRPC dalam pembahasan yang sangat panjang ini

- [Protobuf](#), by Google, BSD-3-Clause License
  - [gRPC](#), by Google, Apache-2.0 License

## C.31. Context: Value, Timeout, & Cancellation

Pada bab ini kita akan belajar pemanfaatan `context.Context` untuk penyisipan dan pembacaan data pada objek `*http.Request`, dan juga untuk handling timeout dan cancelation request.

Sebuah aplikasi web service kecil akan dibuat, tugasnya melakukan pencarian data. Natinya akan dibuat juga middleware `MiddlewareUtility`, tugasnya menyisipkan informasi origin dispatcher request, ke dalam context request, sebelum akhirnya sampai pada handler endpoint yg sedang diakses.

### C.31.1. Context Value

Ok, langsung saja, siapkan folder projek baru dengan struktur seperti berikut.

```
novalagung:1-simple $ tree .
.
├── main.go
└── middleware.go

0 directories, 2 files
```

Pada file `middleware.go` isi dengan `CustomMux` yang pada bab-bab sebelumnya sudah pernah kita gunakan.

```
package main

import "net/http"

type CustomMux struct {
    http.ServeMux
    middlewares []func(next http.Handler) http.Handler
}

func (c *CustomMux) RegisterMiddleware(next func(next http.Handler) http.Handler) {
    c.middlewares = append(c.middlewares, next)
}

func (c *CustomMux) ServeHTTP(w http.ResponseWriter, r *http.Request) {
    var current http.Handler = &c.ServeMux

    for _, next := range c.middlewares {
        current = next(current)
    }

    current.ServeHTTP(w, r)
}
```

Lalu pada file `main.go`, buat satu buah endpoint `/api/search`, dengan isi handler menampilkan data `from` yang diambil dari request context. Data `from` ini di set ke dalam request context oleh middleware `MiddlewareUtility`.

```
package main

import (
    "context"
    "fmt"
    "net/http"
)

type M map[string]interface{}
```

```

func main() {
    mux := new(CustomMux)
    mux.RegisterMiddleware(MiddlewareUtility)

    mux.HandleFunc("/api/search", func(w http.ResponseWriter, r *http.Request) {
        from := r.Context().Value("from").(string)
        w.Write([]byte(from))
    })
}

server := new(http.Server)
server.Handler = mux
server.Addr = ":80"

fmt.Println("Starting server at", server.Addr)
server.ListenAndServe()
}

```

Cara mengakses context request adalah lewat method `.Context()` milik objek request. Lalu chain dengan method `value()` untuk mengambil data sesuai dengan key yang disisipkan pada parameter.

Untuk sekarang, tugas dari endpoint `/api/search` hanya menampilkan data tersebut, tidak lebih.

Selanjutnya siapkan middleware `MiddlewareUtility`. Di dalamnya, ada pengecekan header `Referer`, jika ada maka dijadikan value data `from` (yang kemudian disimpan pada context); sedangkan jika tidak ada maka value-nya berasal dari property `.Host` milik objek request.

```

func MiddlewareUtility(next http.Handler) http.Handler {
    return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
        ctx := r.Context()
        if ctx == nil {
            ctx = context.Background()
        }

        from := r.Header.Get("Referer")
        if from == "" {
            from = r.Host
        }

        ctx = context.WithValue(ctx, "from", from)

        requestWithContext := r.WithContext(ctx)
        next.ServeHTTP(w, requestWithContext)
    })
}

```

Objek request context bisa didapat lewat pengaksesan method `.Context()` milik `*http.Request`. Objek ini bertipe `context.Context` dengan zero type adalah `nil`.

Pada kode di atas, jika context adalah `nil`, maka diinisialisasi dengan context baru lewat `context.Background()`.

Selain lewat `context.Background()`, pembuatan context juga bisa dilakukan lewat `context.TODO()`. Mengenai perbedaannya silakan cek [di laman dokumentasi context](#).

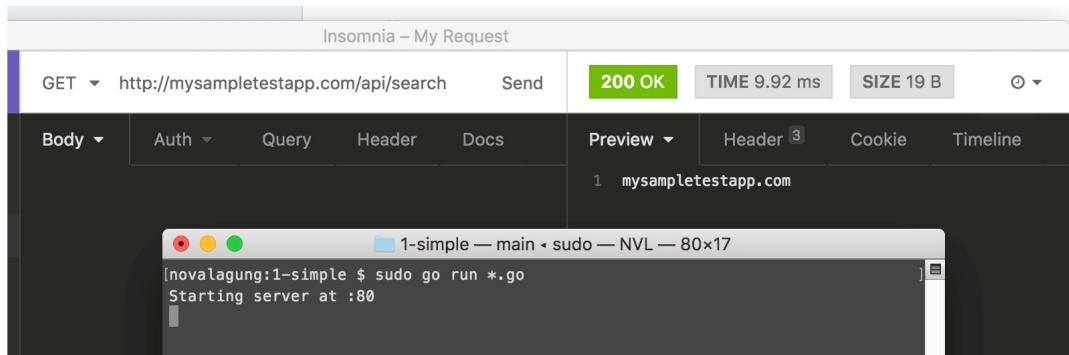
Objek `ctx` yang merupakan `context.Context`, kita tempeli data `from`. Cara melakukannya dengan memanggil statement `contextWithValue()` dengan disisipi 3 buah parameter.

1. Parameter ke-1, isinya adalah objek context.
2. Parameter ke-2, isinya key dari data yang akan disimpan.
3. Parameter ke-3, adalah value data yang akan disimpan.

Fungsi `.WithValue()` di atas mengembalikan objek context, isinya adalah objek context yang disisipkan di parameter pertama pemanggilan fungsi, tapi sudah disisipi data dengan key dari parameter ke-2 dan value dari parameter ke-3. Jadi tampung saja objek context kembalian statement ini ke objek yang sama, yaitu `ctx`.

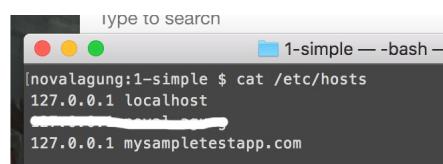
Ok, sekarang objek `ctx` sudah dimodifikasi, objek ini perlu untuk ditempelkan lagi ke objek request. Caranya dengan mengakses method `.WithContext()` milik objek request, lalu gunakan nilai baliknya pada `next.ServeHTTP()`.

Jalankan aplikasi, hasilnya kurang lebih seperti gambar berikut.



O iya, penulis tidak menggunakan `http://localhost` untuk mengakses aplikasi, melainkan menggunakan

`http://mysampletestapp.com`. Domain ini sudah saya arahkan ke 127.0.0.1.



Ok, untuk sekarang sepertinya cukup jelas mengenai penggunaan context pada objek http request. Tinggal kembangkan saja sesuai kebutuhan. Salah satu contoh lainnya, bisa menggunakan context untuk menyimpan data session (yang diambil dari database sesuai dengan session id nya).

## C.31.2. Context Timeout & Cancelation

Kita akan selesaikan program yang sudah dibuat. Nantinya endpoint `/api/search` akan melakukan pencarian ke google sesuai dengan keyword yang diinginkan. Pencarian dilakukan dengan memanfaatkan [Custom Search JSON API](#).

Ubah isi handler endpoint tersebut menjadi seperti berikut.

```
mux.HandleFunc("/api/search", func(w http.ResponseWriter, r *http.Request) {
    ctx := r.Context()

    keyword := r.URL.Query().Get("keyword")
    chanRes := make(chan []byte)
    chanErr := make(chan error)

    go doSearch(ctx, keyword, chanRes, chanErr)

    select {
    case res := <-chanRes:
        w.Header().Set("Content-type", "application/json")
        w.Write(res)
    case err := <-chanErr:
        http.Error(w, err.Error(), http.StatusInternalServerError)
    }
})
```

Proses pencarian dilakukan secara asynchronous lewat fungsi `doSearch()` yang nantinya akan kita buat. Pemanggilannya menggunakan keyword `go` dan disisipkan beberapa parameter yang dua diantaranya bertipe channel.

- Channel `chanRes`, digunakan jika proses pencarian sukses. Data hasil pencarian dilempar ke main routine lewat channel ini, untuk kemudian diteruskan sebagai response endpoint
- Channel `chanErr`, digunakan untuk pass objek error, jika memang terjadi error.

Ok, lanjut, siapkan dua buah konstanta baru.

- Konstanta `SEARCH_MAX_DURATION`, digunakan untuk menge-set max response time. Jika melebihi, maka request langsung di-cancel.

```
var SEARCH_MAX_DURATION = 4 * time.Second
```

- Konstanta `GOOGLE_SEARCH_API_KEY`, ini merupakan API key yang diperlukan dalam penggunaan Custom Search API. Silakan merujuk ke laman [Google Cloud Platform](#) untuk mendapatkan API key.

```
var GOOGLE_SEARCH_API_KEY = "ASSVnHfjd_1tXXXXSyB6WWWWWWveMFgE"
```

Sekarang kita mulai masuk ke bagian pembuatan fungsi pencarian `doSearch()`. Silakan tulis kode berikut.

```
func doSearch(
    ctx context.Context,
    keyword string,
    chanRes chan []byte,
    chanErr chan error,
) {
    innerChanRes := make(chan []byte)
    innerChanErr := make(chan error)

    url := "https://www.googleapis.com/customsearch/v1"
    url = fmt.Sprintf("%s?key=%s", url, GOOGLE_SEARCH_API_KEY)
    url = fmt.Sprintf("%s&cx=017576662512468239146:omuauf_lfve", url)
    url = fmt.Sprintf("%s&callback=hndl", url)
    url = fmt.Sprintf("%s&q=%s", url, keyword)

    from := ctx.Value("from").(string)

    // ...
}
```

Di dalam fungsi tersebut, `url` pencarian dibentuk, data API key dan keyword pencarian disisipkan. Selain itu disiapkan pula `innerChanRes` dan `innerChanErr` yang kegunaannya mirip seperti objek channel yang disisipkan pada pemanggilan fungsi, hanya saja dua channel baru ini digunakan hanya dalam fungsi ini saja.

Lanjut tulis kode berikut.

```
ctx, cancel := context.WithTimeout(ctx, SEARCH_MAX_DURATION)
defer cancel()

req, err := http.NewRequest("GET", url, nil)
if err != nil {
    innerChanErr <- err
    return
}

req = req.WithContext(ctx)
req.Header.Set("Referer", from)
```

```
transport := new(http.Transport)
client := new(http.Client)
client.Transport = transport
```

Objek `ctx` yang di-pass dari luar, dibungkus lagi menggunakan `context.WithTimeout()`. Fungsi ini mengembalikan objek context yang sudah ditambahi data deadline. Tepat setelah statement ini dieksekusi, dalam durasi `SEARCH_MAX_DURATION` context akan di-cancel.

Pengesetan deadline context bisa dilakukan lewat `context.WithTimeout()`, atau bisa juga lewat `context.WithDeadline()`. Perbedaannya pada fungs `.WithDeadline()` parameter yang disisipkan bertipe `time.Time`.

Disiapkan juga objek `req` yang merupakan objek request. Objek ini dibungkus menggunakan `req.WithContext()` dengan isi parameter objek `ctx`, menjadikan objek context yang sudah kita buat tertempel pada request ini. Kegunaannya nanti akan dibahas.

Selain itu, data `from` yang didapat dari request context disisipkan sebagai request header pada objek `req`.

Disiapkan juga objek `transport` yang bertipe `*http.Transport` dan objek `client` yang bertipe `*http.Client`.

Setelah ini, tulis kode untuk dispatch request yang sudah dibuat lalu handle response nya. Jalankan proses-nya sebagai goroutine.

```
go func() {
    resp, err := client.Do(req)
    if err != nil {
        innerChanErr <- err
        return
    }

    if resp != nil {
        defer resp.Body.Close()
        resData, err := ioutil.ReadAll(resp.Body)
        if err != nil {
            innerChanErr <- err
            return
        }
        innerChanRes <- resData
    } else {
        innerChanErr <- errors.New("No response")
    }
}()
```

Request di-trigger lewat statement `client.Do(req)`. Jika menghasilkan error, maka kirim informasi errornya ke channel `innerChanErr`. Cek juga objek response hasil request tersebut, jika kosong maka lempar sebuah error ke channel yang sama.

Baca response body, jika tidak ada masalah, lempar result-nya ke channel `innerChanRes`.

Selanjutnya, kita lakukan pengecekan menggunakan teknik `select case` untuk mengetahui channel mana yang menerima data.

```
select {
case res := <-innerChanRes:
    chanRes <- res
    return
case err := <-innerChanErr:
    transport.CancelRequest(req)
    chanErr <- err
    return
case <-ctx.Done():
    transport.CancelRequest(req)
```

```

chanErr <- errors.New("Search process exceed timeout")
return
}

```

Silakan perhatikan kode di atas, kita akan bahas 2 case pertama.

- Jika channel `innerchanRes` mendapatkan kiriman data, maka langsung diteruskan ke channel `chanRes`.
- Jika channel `innerchanErr` mendapatkan kiriman data, maka langsung diteruskan ke channel `chanErr`. Tak lupa cancel request lewat method `transport.CancelRequest(req)`, ini diperlukan karena request gagal.

Untuk case terakhir `case <- ctx.Done()`, penjelasannya agak sedikit panjang. Objek context, memiliki method `ctx.Done()` yang mengembalikan channel. Channel ini akan melewatkkan data jika deadline timeout context-nya terpenuhi. Dan jika itu terjadi, pada kode di atas langsung dikembalikan sebuah error ke channel `chanErr` dengan isi error `Search process exceed timeout`; tak lupa request-nya juga di-cancel.

Request perlu di-cancel karena jika waktu sudah mencapai deadline context (yaitu `SEARCH_MAX_DURATION`), pada saat tersebut bisa saja request belum selesai, maka dari itu request perlu di-cancel.

Jika di-summary, maka yang dilakukan oleh fungsi `doSearch` kurang lebih sebagai berikut sebagai berikut.

- Jika request pencarian tak lebih dari `SEARCH_MAX_DURATION` :
  - Jika hasilnya sukses, maka kembalikan response body lewat channel `chanRes`.
  - Jika hasilnya gagal, maka kembalikan error-nya lewat channel `chanErr`.
- Jika request pencarian lebih dari `SEARCH_MAX_DURATION`, maka dianggap *timeout*, langsung lempar error `timeout` ke channel `chanErr`.

Jalankan, lihat hasilnya.

The screenshot shows a Postman interface with the following details:

- Method: GET
- URL: p.com/api/search?keyword=golang
- Status: 200 OK
- Time: 1.38 s
- Size: 13.8 KB
- Preview tab shows the JSON response:

```

        "totalResults": "2410000",
        "formattedTotalResults": "2,410,000"
      },
      "items": [
        {
          "kind": "customsearch#result",
          "title": "https://play.golang.org/p/3IFJkluUVc https://play.golang.org/p/...",
          "htmlTitle": "https://play.golang.org/p/3IFJkluUVc https://play.golang.org/p/...",
          "link": "http://zoo.cs.yale.edu/classes/cs474/s2018/Examples/cpsc474_20180130.pdf",
          "displayLink": "zoo.cs.yale.edu",
          "snippet": "Jan 30, 2018 ... https://play.golang.org/p/3IFJkluUVc https://play.golang.org/p/Is4evuDNnt. Jan \n30 Page 1. Page 2. Analysis of 1-player Finite Probabilistic ...",
          "htmlSnippet": "Jan 30, 2018 <b><b>golang</b>.org/p/3IFJkluUVc &iddot; https://play.golang.org/p/Is4evuDNnt. Jan <br>\n30 Page 1. Page 2. Analysis of 1-player Finite Probabilistic&ampnbsp...",
          "cacheId": "y3VAjmdWTnAJ",
          "mime": "application/pdf",
          "fileFormat": "PDF/Adobe Acrobat",
          "formattedUrl": "zoo.cs.yale.edu/classes/cs474/s2018/.../cpsc474_20180130.pdf",
          "htmlFormattedUrl": "zoo.cs.yale.edu/classes/cs474/s2018/.../cpsc474_20180130.pdf",
          "pageMap": {
            "metatags": [
              {
                "producer": "Microsoft® OneNote® 2016",
                "creator": "Microsoft® OneNote® 2016",
                "creationDate": "D:20180201164456-05'00",
                "moddate": "D:20180201164456-05'00"
              }
            ]
          }
        },
        ...
      ],
      "searchInformation": {
        "searchTime": "0.001",
        "currentSearchTerms": "golang",
        "useSitelinks": false
      }
    }
  
```

Informasi tambahan: best practice mengenai cancelation context adalah untuk selalu menambahkan `defer cancel()` setelah (cancelation) context dibuat. Lebih detailnya silakan baca <https://blog.golang.org/context>.

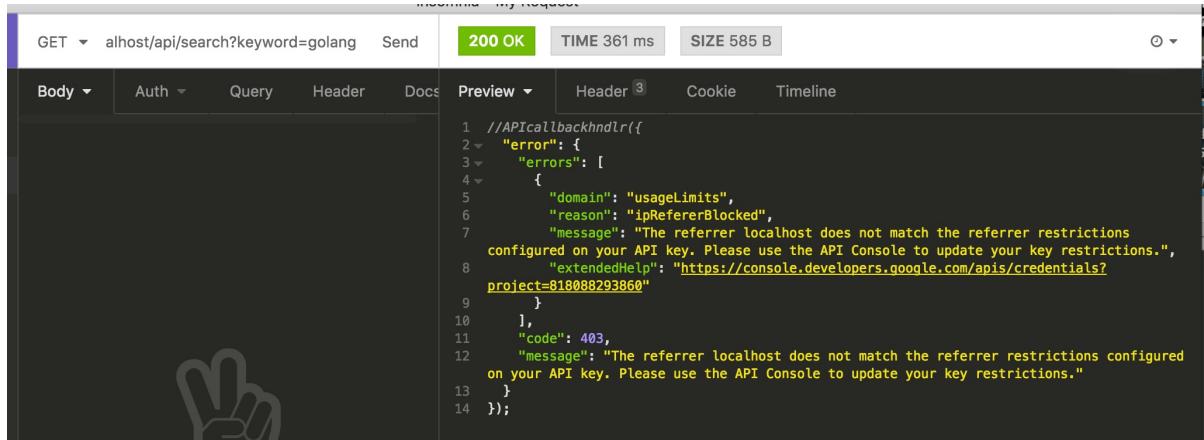
### C.31.3. Google Search API Restrictions Referer

Di bagian awal bab ini, kita belajar mengenai context value. Kenapa penulis memilih menggunakan context untuk menyisipkan data referer, kenapa tidak contoh yg lebih umum seperti session dan lainnya? sebenarnya ada alasannya.

Silakan coba akses kedua url berikut.

- <http://mysampletestapp.com/api/search?keyword=golang>
- <http://localhost/api/search?keyword=golang>

Harusnya yang dikembalikan sama, tapi kenyataannya pengaksesan lewat url `localhost` menghasilkan error.



The screenshot shows a network request in a browser's developer tools. The request is a GET to `localhost/api/search?keyword=golang`. The status is 200 OK, time is 361 ms, and size is 585 B. The response body contains a JSON error message:

```

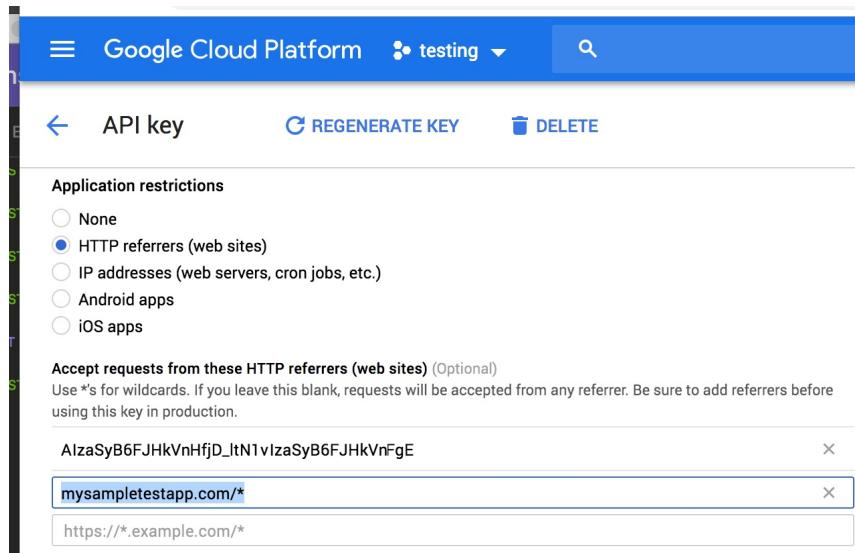
1 //APICallbackndlr{
2   "error": {
3     "errors": [
4       {
5         "domain": "usageLimits",
6         "reason": "ipReferrerBlocked",
7         "message": "The referrer localhost does not match the referrer restrictions
configured on your API key. Please use the API Console to update your key restrictions.",
8         "extendedHelp": "https://console.developers.google.com/apis/credentials?
project=818088293860"
9       },
10      ],
11      "code": 403,
12      "message": "The referrer localhost does not match the referrer restrictions configured
on your API key. Please use the API Console to update your key restrictions."
13    };
14  };

```

Error message:

The referrer localhost does not match the referrer restrictions configured on your API key. Please use the API Console to update your key restrictions.

Error di atas muncul karena, host `localhost` belum didaftarkan pada API console. Berbeda dengan `mysampletestapp.com` yang sudah didaftarkan, host ini berhak mengakses menggunakan API key yang kita gunakan.



## C.32. JSON Web Token (JWT)

Pada bab ini kita akan belajar JSON Web Token (JWT) dan cara penerapannya di bahasa go lang.

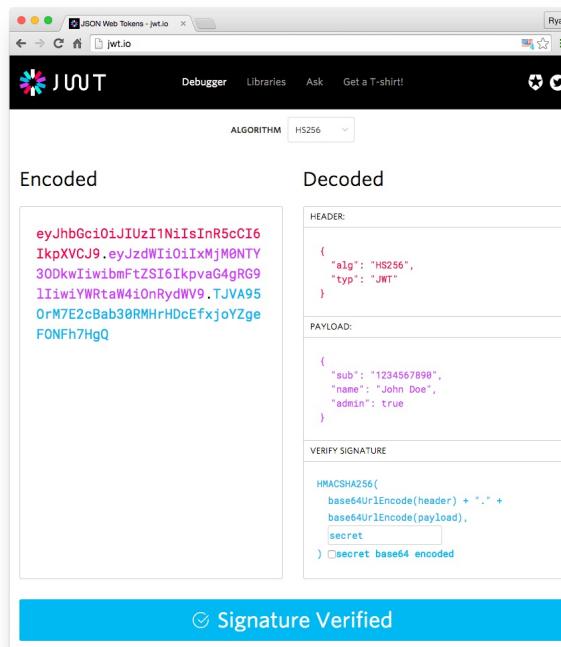
### C.31.1. Definisi

JWT merupakan salah satu standar JSON ([RFC 7519](#)) untuk akses token. Token dibentuk dari kombinasi beberapa informasi yang di-encode dan di-enkripsi. Informasi yang dimaksud adalah header, payload, dan signature.

Contoh JWT:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvvaG4gRG9lIiwiawF0IjoxNTE2MjM5MDIyfQ.Sf1KxwRJSMeKKF2QT4fwpMeJf36P0k6yJV_adQssw5c
```

Skema JWT:



- **Header**, isinya adalah jenis algoritma yang digunakan untuk generate signature.
- **Payload**, isinya adalah data penting untuk keperluan otentikasi, seperti *grant*, *group*, kapan login terjadi, dan atau lainnya. Data ini dalam konteks JWT biasa disebut dengan **CLAIMS**.
- **Signature**, isinya adalah hasil dari enkripsi data menggunakan algoritma kriptografi. Data yang dimaksud adalah gabungan dari (encoded) header, (encoded) payload, dan secret key.

Umumnya pada aplikasi yang menerapkan teknik otorisasi menggunakan token, data token tersebut di-generate di back end secara acak (menggunakan algoritma tertentu) lalu disimpan di database bersamaan dengan data user. Token tersebut tidak ada isinya, hanya random string (atau mungkin saja tidak). Nantinya ketika ada request, token yang disertakan di request dicocokan dengan token yang ada di database, dilanjutkan dengan pengecekan grant/group/sejenisnya, untuk menentukan apakah request tersebut adalah verified request yang memang berhak mengakses endpoint.

Pada aplikasi yang menerapkan JWT, yang terjadi berbeda. Token adalah hasil dari proses kombinasi, encoding, dan enkripsi terhadap beberapa informasi. Nantinya pada sebuah request, pengecekan token tidak dilakukan dengan membandingkan token yang ada di request vs token yang tersimpan di database, karena memang token pada JWT tidak disimpan di database. Pengecekan token dilakukan dengan cara mengecek hasil decode dan decrypt token yang ditautkan dalam request.

Mungkin sekilas terlihat mengerikan, terlihat sangat gampang sekali untuk di-retas, buktinya adalah data otorisasi bisa dengan mudah diambil dari token JWT. Dan penulis sampaikan, bahwa ini adalah presepsi yang salah.

Informasi yang ada dalam token, selain di-encode, juga di-enkripsi. Dalam enkripsi diperlukan private key atau secret key, dan hanya pengembang yang tau. Jadi pastikan simpan baik-baik key tersebut.

Selama peretas tidak tau secret key yang digunakan, hasil decoding dan dekripsi data **PASTI TIDAK VALID**.

## C.32.2. Persiapan Praktek

Kita akan buat sebuah aplikasi RESTful web service sederhana, isinya dua buah endpoint `/index` dan `/login`. Berikut merupakan spesifikasi aplikasinya:

- Pengaksesan `/index` memerlukan token JWT.
- Token didapat dari proses otentifikasi ke endpoint `/login` dengan menyisipkan username dan password.
- Pada aplikasi yang sudah kita buat, sudah ada data list user yang tersimpan di database (sebenarnya bukan di-database, tapi di file json).

Ok, sekarang siapkan folder projek dengan skema seperti berikut:

```
$ tree .
.
├── main.go
├── middleware.go
└── users.json

0 directories, 3 files
```

### File `middleware.go`

Isi file `middleware.go` dengan kode middleware yang sudah biasa kita gunakan.

```
package main

import "net/http"

type CustomMux struct {
    http.ServeMux
    middlewares []func(next http.Handler) http.Handler
}

func (c *CustomMux) RegisterMiddleware(next func(next http.Handler) http.Handler) {
    c.middlewares = append(c.middlewares, next)
}

func (c *CustomMux) ServeHTTP(w http.ResponseWriter, r *http.Request) {
    var current http.Handler = &c.ServeMux

    for _, next := range c.middlewares {
        current = next(current)
    }

    current.ServeHTTP(w, r)
}
```

## File users.json

Juga isi file `users.json` yang merupakan database aplikasi. Silakan tambahkan data JSON berikut.

```
[{
    "username": "noval",
    "password": "kaliparejaya123",
    "email": "terpalmurah@gmail.com",
    "group": "admin"
}, {
    "username": "farhan",
    "password": "masokpakeko",
    "email": "cikrakbaja@gmail.com",
    "group": "publisher"
}]
```

## File main.go

Sekarang kita fokus ke file `main.go`. Import packages yang diperlukan. Salah satu dari packages tersebut adalah [jwt-go](#), yang digunakan untuk keperluan operasi JWT.

```
package main

import (
    "context"
    "encoding/json"
    "fmt"
    "io/ioutil"
    "net/http"
    "os"
    "path/filepath"
    "strings"

    jwt "github.com/dgrijalva/jwt-go"
    "github.com/novalagung/gubrak"
)
```

Masih di file yang sama, siapkan 4 buah konstanta yaitu: nama aplikasi, durasi login, metode enkripsi token, dan secret key.

```
type M map[string]interface{}

var APPLICATION_NAME = "My Simple JWT App"
var LOGIN_EXPIRATION_DURATION = time.Duration(1) * time.Hour
var JWT_SIGNING_METHOD = jwt.SigningMethodHS256
var JWT_SIGNATURE_KEY = []byte("the secret of kalim dor")
```

Kemudian buat fungsi `main()`, siapkan didalamnya sebuah `mux` baru, dan daftarkan middleware `MiddlewareJWTAuthorization` dan dua buah rute `/index` dan `/login`.

```
func main() {
    mux := new(CustomMux)
    mux.RegisterMiddleware(MiddlewareJWTAuthorization)

    mux.HandleFunc("/login", HandlerLogin)
    mux.HandleFunc("/index", HandlerIndex)

    server := new(http.Server)
    server.Handler = mux
```

```

server.Addr = ":8080"

fmt.Println("Starting server at", server.Addr)
server.ListenAndServe()
}

```

Middleware `MiddlewareJWTAuthorization` nantinya akan kita buat, tugasnya memvalidasi setiap request yang masuk, dengan cara mengecek token JWT yang disertakan. Middleware ini hanya berguna pada request ke selain endpoint `/login`, karena pada endpoint tersebut proses otentikasi terjadi.

### C.32.3. Otentikasi & Generate Token

Siapkan handler `HandlerLogin`. Tulis kode berikut.

```

func HandlerLogin(w http.ResponseWriter, r *http.Request) {
    if r.Method != "POST" {
        http.Error(w, "Unsupported http method", http.StatusBadRequest)
        return
    }

    username, password, ok := r.BasicAuth()
    if !ok {
        http.Error(w, "Invalid username or password", http.StatusBadRequest)
        return
    }

    // ...
}

```

Handler ini bertugas untuk meng-otentikasi client/consumer. Data username dan password dikirimkan ke endpoint dalam bentuk **Basic Auth**. Data tersebut kemudian disisipkan dalam pemanggilan fungsi otentikasi `authenticateUser()`, yang nantinya juga akan kita buat.

```

ok, userInfo := authenticateUser(username, password)
if !ok {
    http.Error(w, "Invalid username or password", http.StatusBadRequest)
    return
}

```

Fungsi `authenticateUser()` memiliki dua nilai balik yang ditampung oleh variabel berikut:

- Variabel `ok`, penanda sukses tidaknya otentikasi.
- Variabel `userInfo`, isinya informasi user yang sedang login, datanya didapat dari `data.json` (tetapi tanpa `password`).

Selanjutnya kita akan buat objek claims. Objek ini harus memenuhi persyaratan interface `jwt.Claims`. Objek claims bisa dibuat dari tipe `map` dengan cara membungkusnya dalam fungsi `jwt.MapClaims()`; atau dengan meng-embed interface `jwt.StandardClaims` pada struct baru, dan cara inilah yang akan kita pakai.

Seperti yang sudah kita bahas di awal, bahwa claims isinya adalah data-data untuk keperluan otentikasi. Dalam prakteknya, claims merupakan sebuah objek yang memiliki banyak property atau fields. Nah, objek claims **harus** memiliki fields yang termasuk di dalam list **JWT Standard Fields**. Interface `jwt.StandardClaims` merupakan representasi dari fields yang dimaksud.

Pada aplikasi yang sedang kita kembangkan, claims selain menampung standard fields, juga menampung beberapa informasi lain, oleh karena itu kita perlu buat `struct` baru yang meng-embed `jwt.StandardClaims`.

```
type MyClaims struct {
    jwt.StandardClaims
    Username string `json:"Username"`
    Email    string `json:"Email"`
    Group    string `json:"Group"`
}
```

Ok, struct `MyClaims` sudah siap, sekarang buat objek baru dari struct tersebut.

```
claims := MyClaims{
    StandardClaims: jwt.StandardClaims{
        Issuer:    APPLICATION_NAME,
        ExpiresAt: time.Now().Add(LOGIN_EXPIRATION_DURATION).Unix(),
    },
    Username: userInfo["username"].(string),
    Email:    userInfo["email"].(string),
    Group:    userInfo["group"].(string),
}
```

Ada beberapa standard claims, pada contoh di atas hanya dua yang diisi nilainya, `Issuer` dan `ExpiresAt`, selebihnya kita kosongi. Lalu 3 fields tambahan yang kita buat (`username`, `email`, dan `group`) diisi menggunakan data yang didapat dari `userInfo`.

- `Issuer` (code `iss`), adalah penerbit JWT, dalam konteks ini adalah `APPLICATION_NAME`.
- `ExpiresAt` (code `exp`), adalah kapan token JWT dianggap expired.

Ok, objek `claims` sudah siap, sekarang buat token baru. Pembuatannya dilakukan menggunakan fungsi `jwt.NewWithClaims()` yang menghasilkan nilai balik bertipe `jwt.Token`. Parameter pertama adalah metode enkripsi yang digunakan, yaitu `JWT_SIGNING_METHOD`, dan parameter kedua adalah `claims`.

```
token := jwt.NewWithClaims(
    JWT_SIGNING_METHOD,
    claims,
)
```

Kemudian tanda-tangani token tersebut menggunakan secret key yang sudah didefinisikan di `JWT_SIGNATURE_KEY`, caranya dengan memanggil method `SignedString()` milik objek `jwt.Token`. Pemanggilan method ini mengembalikan data token string yang kemudian dijadikan nilai balik handler. Token string inilah yang dibutuhkan client/consumer untuk bisa mengakses endpoints yang ada.

```
signedToken, err := token.SignedString(JWT_SIGNATURE_KEY)
if err != nil {
    http.Error(w, err.Error(), http.StatusBadRequest)
    return
}

tokenString, _ := json.Marshal(M{ "token": signedToken })
w.Write([]byte(tokenString))
```

Bagian otentifikasi dan generate token sebenarnya cukup sampai disini. Tapi sebenarnya ada yang kurang, yaitu fungsi `authenticateUser()`. Silakan buat fungsi tersebut.

```
func authenticateUser(username, password string) (bool, M) {
    basePath, _ := os.Getwd()
    dbPath := filepath.Join(basePath, "users.json")
    buf, _ := ioutil.ReadFile(dbPath)

    data := make([]M, 0)
    err := json.Unmarshal(buf, &data)
```

```

if err != nil {
    return false, nil
}

res, _ := gubrak.Find(data, func(each M) bool {
    return each["username"] == username && each["password"] == password
})

if res != nil {
    resM := res.(M)
    delete(resM, "password")
    return true, resM
}

return false, nil
}

```

Isi fungsi `authenticateUser()` cukup jelas, sesuai namanya, yaitu melakukan pencocokan username dan password dengan data yang ada di dalam file `users.json`.

## C.32.4. JWT Authorization Middleware

Sekarang kita perlu menyiapkan `MiddlewareJWTAuthorization`, yang tugasnya adalah mengecek setiap request yang masuk ke endpoint selain `/login`, apakah ada akses token yang dibawa atau tidak. Dan jika ada, akan diverifikasi valid tidaknya token tersebut.

```

func MiddlewareJWTAuthorization(next http.Handler) http.Handler {
    return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {

        if r.URL.Path == "/login" {
            next.ServeHTTP(w, r)
            return
        }

        authorizationHeader := r.Header.Get("Authorization")
        if !strings.Contains(authorizationHeader, "Bearer") {
            http.Error(w, "Invalid token", http.StatusBadRequest)
            return
        }

        tokenString := strings.Replace(authorizationHeader, "Bearer ", "", -1)

        // ...
    })
}

```

Kita gunakan skema header `Authorization: Bearer <token>`, sesuai spesifikasi [RFC 6750](#) untuk keperluan penempatan dan pengambilan token.

Token di-extract dari header, kemudian diparsing dan di-validasi menggunakan fungsi `jwt.Parse()`.

```

token, err := jwt.Parse(tokenString, func(token *jwt.Token) (interface{}, error) {
    if method, ok := token.Method.(*jwt.SigningMethodHMAC); !ok {
        return nil, fmt.Errorf("Signing method invalid")
    } else if method != JWT_SIGNING_METHOD {
        return nil, fmt.Errorf("Signing method invalid")
    }

    return JWT_SIGNATURE_KEY, nil
})
if err != nil {
    http.Error(w, err.Error(), http.StatusBadRequest)
}

```

```
    return
}
```

Parameter ke-2 fungsi `jwt.Parse()` berisikan callback untuk pengecekan valid tidaknya signing method, jika valid maka secret key dikembalikan. Fungsi `jwt.Parse()` ini sendiri mengembalikan objek token.

Dari objek token, informasi claims diambil, lalu dilakukan pengecekan valid-tidaknya claims tersebut.

```
claims, ok := token.Claims.(jwt.MapClaims)
if !ok || !token.Valid {
    http.Error(w, err.Error(), http.StatusBadRequest)
    return
}
```

O iya, mungkin ada pertanyaan kenapa objek claims yang dihasilkan `jwt.Parse()` tipenya bukan `MyClaims`. Hal ini karena setelah objek claims dimasukan dalam proses pembentukan token, lewat fungsi `jwt.NewWithClaims()`, objek akan di-encode ke tipe `jwt.MapClaims`.

Data claims yang didapat disisipkan ke dalam context, agar nantinya di endpoint, informasi `userInfo` bisa diambil dengan mudah dari context request.

```
ctx := context.WithValue(context.Background(), "userInfo", claims)
r = r.WithContext(ctx)

next.ServeHTTP(w, r)
```

## Handler Index

Terakhir, kita perlu menyiapkan handler untuk rute `/index`.

```
func HandlerIndex(w http.ResponseWriter, r *http.Request) {
    userInfo := r.Context().Value("userInfo").(jwt.MapClaims)
    message := fmt.Sprintf("hello %s (%s)", userInfo["Username"], userInfo["Group"])
    w.Write([]byte(message))
}
```

Informasi `userInfo` diambil dari context, lalu ditampilkan sebagai response endpoint.

## C.32.5. Testing

Jalankan aplikasi, lalu test menggunakan curl.

### Otentikasi

```
curl -X POST --user noval:kaliparejaya123 http://localhost:8080/login
```

Output:



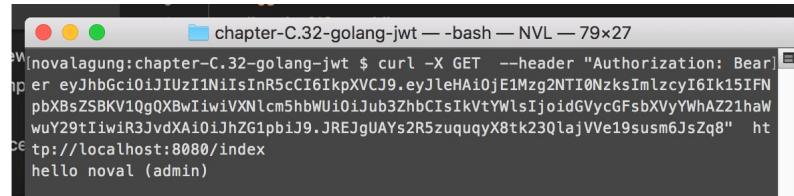
```
[noval@alugung:chapter-C.32-golang-jwt $ curl -X POST --user noval:kaliparejaya123 http://localhost:8080/login
{"token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOiE1Mzg2NTI0NzksImlzcyI6
Ik15IFNpbXBsZSBKV1QgQXBwIiwiVXNlcm5hbWUiOiJub3ZhbCIsIkVtYWlsIjoidGVycGFsbXVyYWh
q8"}]
```

## Mengakses Endpoint

Test endpoint `/index`. Sisipkan token yang dikembalikan pada saat otentikasi, sebagai value header otorisasi dengan skema `Authorization: Bearer <token>`.

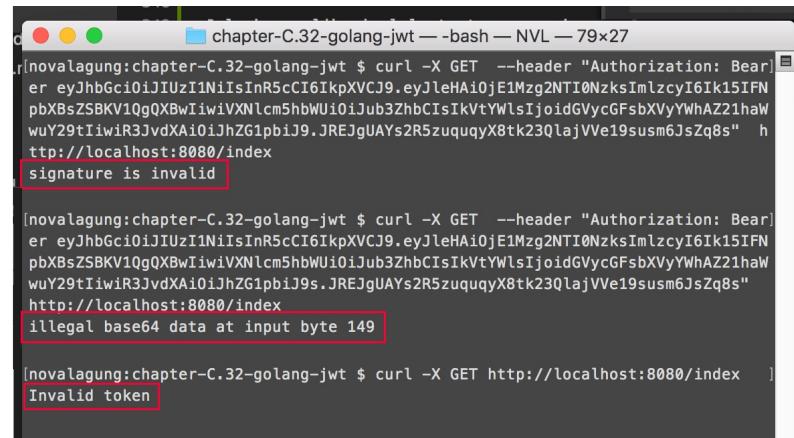
```
curl -X GET \
  --header "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOiE1Mzg2NTI0NzksImlzcyI6Ik15IFN
  NpbXBsZSBKV1QgQXBwIiwiVXNlc5hbWUiOjJub3ZhbcIsIkVtYWlsIjoidGVycGFsbXVyYWhAZ21haWwuY29tIiwiR3JvdXAiOjJhZG1pbij9.JREJgUAYs2R5zuquqyX8tk23QlajVVe19susm6JsZq8" \
  http://localhost:8080/index
```

Output:



```
[novalagung:chapter-C.32-golang-jwt $ curl -X GET --header "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOiE1Mzg2NTI0NzksImlzcyI6Ik15IFN
  NpbXBsZSBKV1QgQXBwIiwiVXNlc5hbWUiOjJub3ZhbcIsIkVtYWlsIjoidGVycGFsbXVyYWhAZ21haWwuY29tIiwiR3JvdXAiOjJhZG1pbij9.JREJgUAYs2R5zuquqyX8tk23QlajVVe19susm6JsZq8" \
  http://localhost:8080/index
hello noval (admin)
```

Semua berjalan sesuai harapan. Agar lebih meyakinkan, coba lakukan beberapa test dengan skenario yg salah, seperti:



```
[novalagung:chapter-C.32-golang-jwt $ curl -X GET --header "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOiE1Mzg2NTI0NzksImlzcyI6Ik15IFN
  NpbXBsZSBKV1QgQXBwIiwiVXNlc5hbWUiOjJub3ZhbcIsIkVtYWlsIjoidGVycGFsbXVyYWhAZ21haWwuY29tIiwiR3JvdXAiOjJhZG1pbij9.JREJgUAYs2R5zuquqyX8tk23QlajVVe19susm6JsZq8s" \
  http://localhost:8080/index
signature is invalid

[novalagung:chapter-C.32-golang-jwt $ curl -X GET --header "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOiE1Mzg2NTI0NzksImlzcyI6Ik15IFN
  NpbXBsZSBKV1QgQXBwIiwiVXNlc5hbWUiOjJub3ZhbcIsIkVtYWlsIjoidGVycGFsbXVyYWhAZ21haWwuY29tIiwiR3JvdXAiOjJhZG1pbij9s.JREJgUAYs2R5zuquqyX8tk23QlajVVe19susm6JsZq8s"
  http://localhost:8080/index
illegal base64 data at input byte 149

[novalagung:chapter-C.32-golang-jwt $ curl -X GET http://localhost:8080/index
Invalid token]
```