


Go-Lang Web

Eko Kurniawan Khannedy

Eko Kurniawan Khannedy

- Technical architect at one of the biggest ecommerce company in Indonesia
- 10+ years experiences
- www.programmerzamannow.com
- youtube.com/c/ProgrammerZamanNow



Eko Kurniawan Khannedy

- Telegram : [@khannedy](https://t.me/khannedy)
- Facebook : [fb.com/ProgrammerZamanNow](https://www.facebook.com/ProgrammerZamanNow)
- Instagram : [instagram.com/programmerzamannow](https://www.instagram.com/programmerzamannow)
- Youtube : [youtube.com/c/ProgrammerZamanNow](https://www.youtube.com/c/ProgrammerZamanNow)
- Telegram Channel : t.me/ProgrammerZamanNow
- Email : echo.khannedy@gmail.com

Sebelum Belajar

- Go-Lang Dasar
- Go-Lang Modules
- Go-Lang Unit Test
- Go-Lang Goroutines
- Go-Lang Embed
- HTML
- CSS
- JavaScript

Agenda

- Pengenalan Web
- Client dan Server
- Pengenalan Go-Lang Web
- Package net/http
- Membuat Server
- Membuat ServerMux
- File Server
- Middleware
- Dan lain-lain

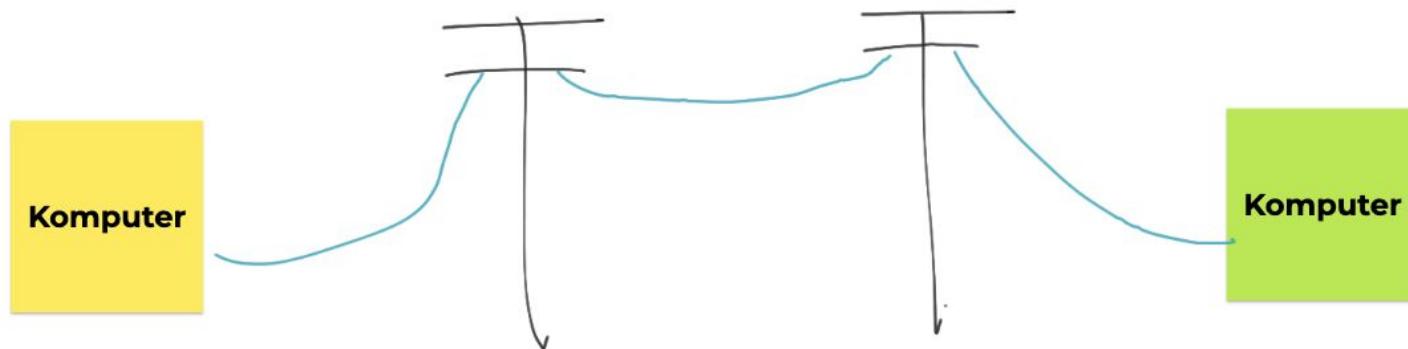
Pengenalan Web

Kenapa Web?

- Saat ini web digunakan oleh jutaan, bahkan mungkin milyaran orang setiap hari
- Dengan web, kita bisa melakukan belajar online, mendengarkan musik online, nonton video online, belanja online, sampai memesan makanan secara online
- Namun perlu diperhatikan, Web bukanlah Internet

Internet

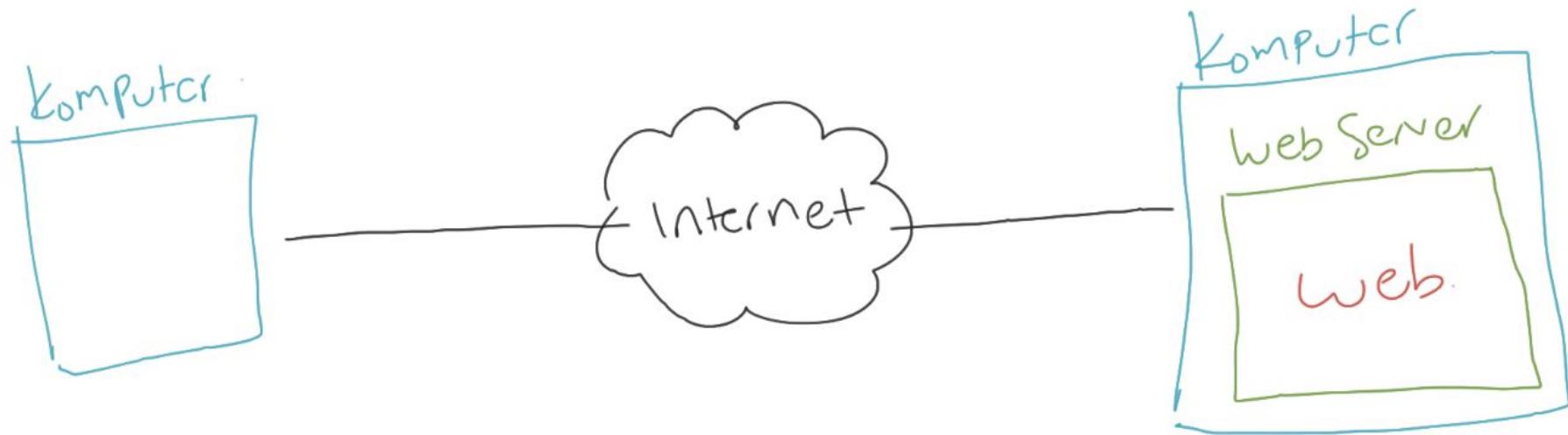
- Internet adalah mekanisme komunikasi antar komputer
- Awal internet ada, untuk komunikasi antar komputer, kita membutuhkan jaringan kabel telepon
- Namun sekarang, semenjak berjamurnya jaringan wifi dan sejenisnya, komunikasi antar komputer menjadi lebih cepat dan mudah



Web

- Web merupakan kumpulan informasi yang tersedia dalam sebuah komputer yang terkoneksi secara terus menerus melalui internet
- Web bisa berisi informasi dalam bentuk apapun, seperti teks, gambar, audio, video dan lain-lain
- Web berjalan di aplikasi yang bernama Web Server, yaitu aplikasi yang digunakan untuk menyimpan dan menyampaikan isi informasi Web

Diagram Web



Web Host

- Pemilik Web, biasanya tidak menjalankan aplikasi Web Server di komputer pribadi nya
- Biasanya mereka akan menyewa komputer di tempat penyedia data center (kumpulan komputer) yang terjamin keandalan dan kecepatan koneksi internetnya
- Pihak penyedia komputer untuk Web Server biasa disebut Web Host



Domain

- Saat komputer Web terhubung ke internet, biasanya dia memiliki alamat
- Alamat ini bernama ip address, formatnya misal nya 172.217.194.94
- Karena alamat ip address sangat menyulitkan untuk diingat
- Untung saja ada yang namanya nama domain
- Nama domain adalah alamat yang bisa digunakan sebagai alias ke ip address
- Misal seperti google.co.id, blibli.com, dan lain-lain
- Dengan nama domain, sebagai manusia kita akan mudah mengingat dibandingkan ip address
- Namun, saat kita menggunakan nama domain, sebenarnya komputer tetap akan mengakses web menggunakan alamat ip address

Web Browser

- Jika Web Server adalah aplikasi yang digunakan untuk menyimpan informasi Web
- Web Browser adalah aplikasi yang digunakan untuk mengakses Web melalui internet
- Kita bisa saja mengakses Web secara langsung tanpa bantuan Web Browser, namun Web Server hanya akan memberikan informasi bahasa mesin seperti HTML, JavaScript, CSS, Gambar, Video dan lain-lain
- Dengan menggunakan Web Browser, semua bahasa mesin tersebut bisa ditampilkan secara visual sehingga kita bisa menyerap informasinya dengan lebih mudah

Client dan Server

Client dan Server

- Web adalah aplikasi berbasis Client dan Server, sekarang pertanyaannya, apa itu Client dan Server?
- Sederhananya client server merupakan konsep arsitektur aplikasi yang menghubungkan dua pihak, sistem client dan sistem server
- Sistem client dan sistem server yang saling berkomunikasi melalui jaringan komputer, internet, atau juga bisa di komputer yang sama

Diagram Client dan Server



Tugas Client dan Server

- Aplikasi Client bertugas mengirim request ke Server, dan menerima response dari Server
- Sedangkan, aplikasi Server bertugas menerima request dari Client, memproses data, dan mengembalikan hasil proses data ke Client

Keuntungan Client dan Server

- Perubahan aplikasi bisa dilakukan dengan mudah di server, tanpa harus membuat perubahan di client, apalagi jika client nya di lokasi yang sulit dijangkau
- Bisa digunakan oleh banyak client pada saat yang bersamaan, walaupun server tidak banyak
- Bisa diakses dari mana saja, asal terhubung satu jaringan dengan server

Contoh Client dan Server

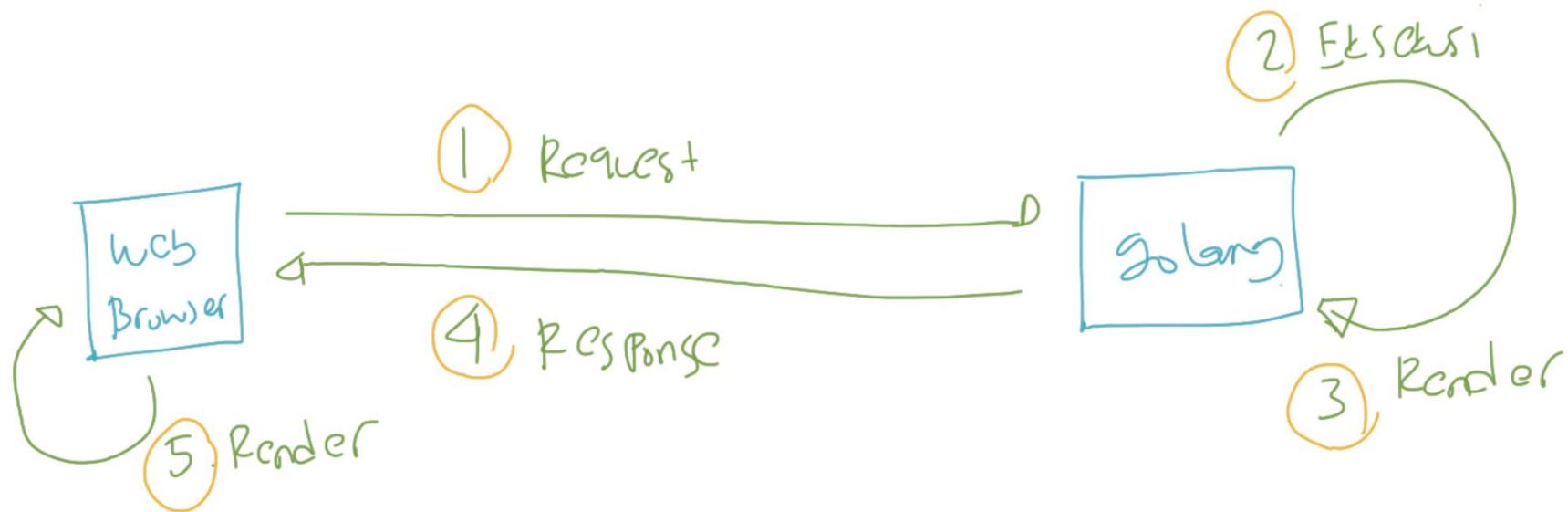
- Web adalah salah satu contoh arsitektur client server
- Aplikasi yang bertugas sebagai Client adalah Web Browser (Chrome, Firefox, Opera, Edge dan lain-lain)
- Aplikasi yang bertugas sebagai Server adalah Web Server, dimana di dalam web server terdapat kode program Web kita

Go-Lang Web

Go-Lang Web

- Go-Lang saat ini populer dijadikan salah satu pilihan bahasa pemrograman untuk membuat Web, terutama Web API (Backend)
- Selain itu, di Go-Lang juga sudah disediakan package untuk membuat Web, bahkan di sertakan pula package untuk implementasi unit testing untuk Web
- Hal ini menjadikan pembuatan Web menggunakan Go-Lang lebih mudah, karena tidak butuh menggunakan library atau framework

Diagram Cara Kerja Go-Lang Web



Cara Kerja Lang Web

1. Web Browser akan melakukan HTTP Request ke Web Server
2. Golang menerima HTTP Request, lalu mengeksekusi request tersebut sesuai dengan yang diminta.
3. Setelah melakukan eksekusi request, Golang akan mengembalikan data dan di render sesuai dengan kebutuhannya, misal HTML, CSS, JavaScript dan lain-lain
4. Golang akan mengembalikan content hasil render tersebut tersebut sebagai HTTP Response ke Web Browser
5. Web Browser menerima content dari Web Server, lalu me-render content tersebut sesuai dengan tipe content nya

Package net/http

- Pada beberapa bahasa pemrograman lain, seperti Java misalnya, untuk membuat web biasanya dibutuhkan tambahan library atau framework
- Sedangkan di Go-Lang sudah disediakan package untuk membuat web bernama package net/http
- Sehingga untuk membuat web menggunakan Go-Lang, kita tidak butuh lagi library tambahan, kita bisa menggunakan package yang sudah tersedia
- Walaupun memang saat kita akan membuat web dalam skala besar, direkomendasikan menggunakan framework karena beberapa hal sudah dipermudah oleh web framework
- Namun pada course ini, kita akan fokus menggunakan package net/http untuk membuat web nya, karena semua framework web Go-Lang akan menggunakan net/http sebagai basis dasar framework nya

Server

Server

- Server adalah struct yang terdapat di package net/http yang digunakan sebagai representasi Web Server di Go-Lang
- Untuk membuat web, kita wajib membuat Server
- Saat membuat data Server, ada beberapa hal yang perlu kita tentukan, seperti host dan juga port tempat dimana Web kita berjalan
- Setelah membuat Server, kita bisa menjalankan Server tersebut menggunakan function ListenAndServe()

Kode : Server

```
server := http.Server{  
    Addr: "localhost:8080",  
}  
  
err := server.ListenAndServe()  
if err != nil {  
    panic(err)  
}
```

Handler



Handler

- Server hanya bertugas sebagai Web Server, sedangkan untuk menerima HTTP Request yang masuk ke Server, kita butuh yang namanya Handler
- Handler di Go-Lang di representasikan dalam interface, dimana dalam kontraknya terdapat sebuah function bernama ServeHTTP() yang digunakan sebagai function yang akan di eksekusi ketika menerima HTTP Request

HandlerFunc

- Salah satu implementasi dari interface Handler adalah HandlerFunc
- Kita bisa menggunakan HandlerFunc untuk membuat function handler HTTP



Kode : Hello World Web

```
var handler http.HandlerFunc = func(writer http.ResponseWriter, request *http.Request) {
    fmt.Fprint(writer, "Hello World")
}

server := http.Server{
    Addr:      "localhost:8080",
    Handler:   handler,
}
err := server.ListenAndServe()
```

ServeMux

ServeMux

- Saat membuat web, kita biasanya ingin membuat banyak sekali endpoint URL
- HandlerFunc sayangnya tidak mendukung itu
- Alternative implementasi dari Handler adalah ServeMux
- ServeMux adalah implementasi Handler yang bisa mendukung multiple endpoint



Kode : ServeMux

```
mux := http.NewServeMux()
mux.HandleFunc("/", func(writer http.ResponseWriter, request *http.Request) {
    fmt.Fprint(writer, "Hello World")
})
mux.HandleFunc("/hi", func(writer http.ResponseWriter, request *http.Request) {
    fmt.Fprint(writer, "Hi")
})
server := http.Server{
    Addr:      "localhost:8080",
    Handler: mux,
}
```

URL Pattern

- URL Pattern dalam ServeMux sederhana, kita tinggal menambahkan string yang ingin kita gunakan sebagai endpoint, tanpa perlu memasukkan domain web kita
- Jika URL Pattern dalam ServeMux kita tambahkan di akhirnya dengan garis miring, artinya semua url tersebut akan menerima path dengan awalan tersebut, misal /images/ artinya akan dieksekusi jika endpoint nya /images/, /images/contoh, /images/contoh/lagi
- Namun jika terdapat URL Pattern yang lebih panjang, maka akan diprioritaskan yang lebih panjang, misal jika terdapat URL /images/ dan /images/thumbnails/, maka jika mengakses /images/thumbnails/ akan mengakses /images/thumbnails/, bukan /images



Kode : ServeMux URL Pattern

```
mux := http.NewServeMux()
mux.HandleFunc("/images/", func(writer http.ResponseWriter, request *http.Request) {
    fmt.Fprint(writer, "Images")
})
mux.HandleFunc("/imagesthumbnails/", func(writer http.ResponseWriter, request *http.Request) {
    fmt.Fprint(writer, "Thumbnails")
})
```

Request

Request

- Request adalah struct yang merepresentasikan HTTP Request yang dikirim oleh Web Browser
- Semua informasi request yang dikirim bisa kita dapatkan di Request
- Seperti, URL, http method, http header, http body, dan lain-lain



Kode : Request

```
var handler http.HandlerFunc = func(writer http.ResponseWriter, request *http.Request) {
    fmt.Fprintln(writer, request.Method)
    fmt.Fprintln(writer, request.RequestURI)
}

server := http.Server{
    Addr:      "localhost:8080",
    Handler:   handler,
}
```

HTTP Test

HTTP Test

- Go-Lang sudah menyediakan package khusus untuk membuat unit test terhadap fitur Web yang kita buat
- Semuanya ada di dalam package net/http/httptest <https://golang.org/pkg/net/http/httptest/>
- Dengan menggunakan package ini, kita bisa melakukan testing handler web di Go-Lang tanpa harus menjalankan aplikasi web nya
- Kita bisa langsung fokus terhadap handler function yang ingin kita test

httptest.NewRequest()

- `NewRequest(method, url, body)` merupakan function yang digunakan untuk membuat `http.Request`
- Kita bisa menentukan method, url dan body yang akan kita kirim sebagai simulasi unit test
- Selain itu, kita juga bisa menambahkan informasi tambahan lainnya pada request yang ingin kita kirim, seperti header, cookie, dan lain-lain

httptest.NewRecorder()

- `httptest.NewRecorder()` merupakan function yang digunakan untuk membuat ResponseRecorder
- ResponseRecorder merupakan struct bantuan untuk merekam HTTP response dari hasil testing yang kita lakukan



Kode : HTTP Test

```
func HelloHandler(writer http.ResponseWriter, request *http.Request) {
    fmt.Fprintln(writer, "Hello World")
}

func TestHelloHandler(t *testing.T) {
    request := httptest.NewRequest("GET", "http://localhost:8080/hello", nil)
    recorder := httptest.NewRecorder()

    HelloHandler(recorder, request)
}
```



Kode : Mengecek Hasil Test

```
response := recorder.Result()
body, _ := io.ReadAll(response.Body)

fmt.Println(response.StatusCode)
fmt.Println(response.Status)
fmt.Println(string(body))
```

Query Parameter

Query Parameter

- Query parameter adalah salah satu fitur yang biasa kita gunakan ketika membuat web
- Query parameter biasanya digunakan untuk mengirim data dari client ke server
- Query parameter ditempatkan pada URL
- Untuk menambahkan query parameter, kita bisa menggunakan ?nama=value pada URL nya

url.URL

- Dalam parameter Request, terdapat attribute URL yang berisi data url.URL
- Dari data URL ini, kita bisa mengambil data query parameter yang dikirim dari client dengan menggunakan method Query() yang akan mengembalikan map

Kode : Query Parameter

```
func SayHello(writer http.ResponseWriter, request *http.Request) {  
    name := request.URL.Query().Get("name")  
    if name == "" {  
        fmt.Fprint(writer, "Hello")  
    } else {  
        fmt.Fprintf(writer, "Hello %s", name)  
    }  
}
```



Kode : Test Query Parameter

```
request := httptest.NewRequest("GET", "http://localhost:8080/hello?name=Eko", nil)
recorder := httptest.NewRecorder()

SayHello(recorder, request)

response := recorder.Result()
body, _ := io.ReadAll(response.Body)

fmt.Println(string(body))
```

Multiple Query Parameter

- Dalam spesifikasi URL, kita bisa menambahkan lebih dari satu query parameter
- Ini cocok sekali jika kita memang ingin mengirim banyak data ke server, cukup tambahkan query parameter lainnya
- Untuk menambahkan query parameter, kita bisa gunakan tanda & lalu diikuti dengan query parameter berikutnya

Kode : Multiple Query Parameter

```
func MultipleParameter(writer http.ResponseWriter, request *http.Request) {  
    firstName := request.URL.Query().Get("first_name")  
    lastName := request.URL.Query().Get("last_name")  
    fmt.Fprintf(writer, "%s %s", firstName, lastName)  
}
```



Kode : Test Multiple Query Parameter

```
request := httptest.NewRequest("GET", "http://localhost/?first_name=Eko&last_name=Kurniawan", nil)
recorder := httptest.NewRecorder()

MultipleParameter(recorder, request)

response := recorder.Result()
body, _ := io.ReadAll(response.Body)
fmt.Println(string(body))
```

Multiple Value Query Parameter

- Sebenarnya URL melakukan parsing query parameter dan menyimpannya dalam map[string][]string
- Artinya, dalam satu key query parameter, kita bisa memasukkan beberapa value
- Caranya kita bisa menambahkan query parameter dengan nama yang sama, namun value berbeda, misal :
- name=Eko&name=Kurniawan

Kode : Multiple Value Query Parameter

```
func MultipleParameter(writer http.ResponseWriter, request *http.Request) {  
    var query url.Values = request.URL.Query()  
    var names []string = query["name"]  
    fmt.Fprintln(writer, strings.Join(names, ", "))  
}
```



Kode : Test Multiple Value Query Parameter

```
request := httptest.NewRequest("GET", "http://localhost/?name=Eko&name=Kurniawan", nil)
recorder := httptest.NewRecorder()

MultipleValueParameter(recorder, request)

response := recorder.Result()
body, _ := io.ReadAll(response.Body)
fmt.Println(string(body))
```

Header

Header

- Selain Query Parameter, dalam HTTP, ada juga yang bernama Header
- Header adalah informasi tambahan yang biasa dikirim dari client ke server atau sebaliknya
- Jadi dalam Header, tidak hanya ada pada HTTP Request, pada HTTP Response pun kita bisa menambahkan informasi header
- Saat kita menggunakan browser, biasanya secara otomatis header akan ditambahkan oleh browser, seperti informasi browser, jenis tipe content yang dikirim dan diterima oleh browser, dan lain-lain

Request Header

- Untuk menangkap request header yang dikirim oleh client, kita bisa mengambilnya di Request.Header
- Header mirip seperti Query Parameter, isinya adalah map[string][]string
- Berbeda dengan Query Parameter yang case sensitive, secara spesifikasi, Header key tidaklah case sensitive

Kode : Request Header

```
func RequestHeader(writer http.ResponseWriter, request *http.Request) {  
    contentType := request.Header.Get("content-type")  
    fmt.Fprint(writer, contentType)  
}  
}
```



Kode : Test Request Header

```
request := httptest.NewRequest("GET", "http://localhost/", nil)
request.Header.Add("Content-Type", "application/json")
recorder := httptest.NewRecorder()

RequestHeader(recorder, request)

response := recorder.Result()
body, _ := io.ReadAll(response.Body)
fmt.Println(string(body))
```

Response Header

- Sedangkan jika kita ingin menambahkan header pada response, kita bisa menggunakan function ResponseWriter.Header()

Kode : Response Header

```
func ResponseHeader(writer http.ResponseWriter, request *http.Request) {  
    writer.Header().Add("X-Powered-By", "Programmer Zaman Now")  
    fmt.Fprint(writer, "OK")  
}
```



Kode : Test Response Header

```
request := httptest.NewRequest("GET", "http://localhost/", nil)
recorder := httptest.NewRecorder()
```

```
ResponseHeader(recorder, request)
```

```
poweredBy := recorder.Header().Get("x-powered-by")
fmt.Println(poweredBy)
```

Form Post

Form Post

- Saat kita belajar HTML, kita tahu bahwa saat kita membuat form, kita bisa submit datanya dengan method GET atau POST
- Jika menggunakan method GET, maka hasilnya semua data di form akan menjadi query parameter
- Sedangkan jika menggunakan POST, maka semua data di form akan dikirim via body HTTP request
- Di Go-Lang, untuk mengambil data Form Post sangatlah mudah

Request.PostForm

- Semua data form post yang dikirim dari client, secara otomatis akan disimpan dalam attribute Request.PostForm
- Namun sebelum kita bisa mengambil data di attribute PostForm, kita wajib memanggil method Request.ParseForm() terlebih dahulu, method ini digunakan untuk melakukan parsing data body apakah bisa di parsing menjadi form data atau tidak, jika tidak bisa di parsing, maka akan menyebabkan error

Kode : Form Post

```
func FormPost(writer http.ResponseWriter, request *http.Request) {  
    err := request.ParseForm()  
    if err != nil {  
        panic(err)  
    }  
    firstName := request.PostForm.Get("firstName")  
    lastName := request.PostForm.Get("lastName")  
    fmt.Fprintf(writer, "%s %s", firstName, lastName)  
}
```



Kode : Test Form Post

```
requestBody := strings.NewReader("firstName=Eko&lastName=Kurniawan")
request := httptest.NewRequest("POST", "http://localhost/", requestBody)
request.Header.Add("Content-Type", "application/x-www-form-urlencoded")
recorder := httptest.NewRecorder()

FormPost(recorder, request)

response := recorder.Result()
body, _ := io.ReadAll(response.Body)
fmt.Println(string(body))
```

Response Code

Response Code

- Dalam HTTP, terdapat yang namanya response code
- Response code merupakan representasi kode response
- Dari response code ini kita bisa melihat apakah sebuah request yang kita kirim itu sukses diproses oleh server atau gagal
- Ada banyak sekali response code yang bisa kita gunakan saat membuat web
- <https://developer.mozilla.org/en-US/docs/Web/HTTP/Status>

Mengubah Response Code

- Secara default, jika kita tidak menyebutkan response code, maka response code nya adalah 200 OK
- Jika kita ingin mengubahnya, kita bisa menggunakan function ResponseWriter.WriteHeader(int)
- Semua data status code juga sudah disediakan di Go-Lang, jadi kita ingin, kita bisa gunakan variable yang sudah disediakan : <https://github.com/golang/go/blob/master/src/net/http/status.go>

Kode : Response Code

```
func ResponseCode(writer http.ResponseWriter, request *http.Request) {  
    name := request.URL.Query().Get("name")  
    if name == "" {  
        writer.WriteHeader(400) // BadRequest  
        fmt.Fprint(writer, "name is empty")  
    } else {  
        writer.WriteHeader(200)  
        fmt.Sprintf(writer, "Hi %s", name)  
    }  
}
```



Kode : Test Response Code

```
request := httptest.NewRequest("GET", "http://localhost/", nil)
recorder := httptest.NewRecorder()

ResponseCode(recorder, request)

response := recorder.Result()
body, _ := io.ReadAll(response.Body)
fmt.Println(response.StatusCode)
fmt.Println(response.Status)
fmt.Println(string(body))
```

Cookie

Stateless

- HTTP merupakan stateless antara client dan server, artinya server tidak akan menyimpan data apapun untuk mengingat setiap request dari client
- Hal ini bertujuan agar mudah melakukan scalability di sisi server
- Lantas bagaimana caranya agar server bisa mengingat sebuah client? Misal ketika kita sudah login di website, server otomatis harus tahu jika client tersebut sudah login, sehingga request selanjutnya, tidak perlu diminta untuk login lagi
- Untuk melakukan hal ini, kita bisa memanfaatkan Cookie

Cookie

- Cookie adalah fitur di HTTP dimana server bisa memberi response cookie (key-value) dan client akan menyimpan cookie tersebut di web browser
- Request selanjutnya, client akan selalu membawa cookie tersebut secara otomatis
- Dan server secara otomatis akan selalu menerima data cookie yang dibawa oleh client setiap kalo client mengirimkan request

Membuat Cookie

- Cookie merupakan data yang dibuat di server dan sengaja agar disimpan di web browser
- Untuk membuat cookie di server, kita bisa menggunakan function `http.SetCookie()`

Kode : Membuat Cookie

```
func SetCookie(writer http.ResponseWriter, request *http.Request) {  
    cookie := new(http.Cookie)  
    cookie.Name = "X-PZN-Name"  
    cookie.Value = request.URL.Query().Get("name")  
    cookie.Path = "/"  
  
    http.SetCookie(writer, cookie)  
    fmt.Fprint(writer, "Success Create Cookie")  
}
```

Kode : Mengambil Cookie

```
func GetCookie(writer http.ResponseWriter, request *http.Request) {  
    cookie, err := request.Cookie("X-PZN-Name")  
    if err != nil {  
        fmt.Fprint(writer, "No Cookie")  
    } else {  
        fmt.Sprintf(writer, "Hello %s", cookie.Value)  
    }  
}
```



Kode : Mencoba Cookie

```
mux := http.NewServeMux()
mux.HandleFunc("/set-cookie", SetCookie)
mux.HandleFunc("/get-cookie", GetCookie)

server := http.Server{
    Addr:      "localhost:8080",
    Handler:   mux,
}

err := server.ListenAndServe()
if err != nil {
    panic(err)
}
```



Kode : Test Membuat Cookie

```
request := httptest.NewRequest(http.MethodGet, "http://localhost/?name=Eko", nil)
recorder := httptest.NewRecorder()

SetCookie(recorder, request)

cookies := recorder.Result().Cookies()

for _, cookie := range cookies {
    fmt.Printf("%s : %s\n", cookie.Name, cookie.Value)
}
```

Kode : Test Mengambil Cookie

```
request := httptest.NewRequest(http.MethodGet, "http://localhost/", nil)
cookie := new(http.Cookie)
cookie.Name = "X-PZN-Name"
cookie.Value = "Eko"
request.AddCookie(cookie)

recorder := httptest.NewRecorder()

GetCookie(recorder, request)

response := recorder.Result()
body, _ := io.ReadAll(response.Body)
fmt.Println(string(body))
```

FileServer

FileServer

- Go-Lang memiliki sebuah fitur yang bernama FileServer
- Dengan ini, kita bisa membuat Handler di Go-Lang web yang digunakan sebagai static file server
- Dengan menggunakan FileServer, kita tidak perlu manual me-load file lagi
- FileServer adalah Handler, jadi bisa kita tambahkan ke dalam http.Server atau http.ServeMux



Kode : FileServer

```
directory := http.Dir("./resources")
fileServer := http.FileServer(directory)

mux := http.NewServeMux()
mux.Handle("/static/", fileServer)

server := http.Server{
    Addr:    "localhost:8080",
    Handler: mux,
}
```

404 Not Found

- Jika kita coba jalankan, saat kita membuka misal /static/index.js, maka akan dapat error 404 Not Found
- Kenapa ini terjadi?
- Hal ini dikarenakan FileServer akan membaca url, lalu mencari file berdasarkan url nya, jadi jika kita membuat /static/index.js, maka FileServer akan mencari ke file /resources/static/index.js
- Hal ini menyebabkan 404 Not Found karena memang file nya tidak bisa ditemukan
- Oleh karena itu, kita bisa menggunakan function http.StripPrefix() untuk menghapus prefix di url



Kode : FileServer dengan StripPrefix

```
directory := http.Dir("./resources")
fileServer := http.FileServer(directory)

mux := http.NewServeMux()
mux.Handle("/static/", http.StripPrefix("/static", fileServer))

server := http.Server{
    Addr:     "localhost:8080",
    Handler: mux,
}
```

Go-Lang Embed

- Di Go-Lang 1.16 terdapat fitur baru yang bernama Go-Lang embed
- Dalam Go-Lang embed kita bisa embed file ke dalam binary distribution file, hal ini mempermudah sehingga kita tidak perlu meng-copy static file lagi
- Go-Lang Embed juga memiliki fitur yang bernama embed.FS, fitur ini bisa diintegrasikan dengan FileServer



Kode : FileServer Go-Lang Embed

```
//go:embed resources
var resources embed.FS

func TestFileServerGoEmbed(t *testing.T) {
    fileServer := http.FileServer(http.FS(resources))

    mux := http.NewServeMux()
    mux.Handle("/static/", http.StripPrefix("/static", fileServer))

    server := http.Server{
        Addr:     "localhost:8080",
        Handler: mux,
```

404 Not Found

- Jika kita coba jalankan, dan coba buka /static/index.js, maka kita akan mendapatkan error 404 Not Found
- Kenapa ini terjadi? Hal ini karena di Go-Lang embed, nama folder ikut menjadi nama resource nya, misal resources/index.js, jadi untuk mengaksesnya kita perlu gunakan URL /static/resources/index.js
- Jika kita ingin langsung mengakses file index.js tanpa menggunakan resources, kita bisa menggunakan function fs.Sub() untuk mendapatkan sub directory

Kode : FileServer Go-Lang Embed

```
//go:embed resources
var resources embed.FS

func TestFileServerGoEmbed(t *testing.T) {
    directory, _ := fs.Sub(resources, "resources")
    fileServer := http.FileServer(http.FS(directory))

    mux := http.NewServeMux()
    mux.Handle("/static/", http.StripPrefix("/static", fileServer))

    server := http.Server{
        Addr:     "localhost:8080",
        Handler: mux,
    }
```

ServeFile

ServeFile

- Kadang ada kasus misal kita hanya ingin menggunakan static file sesuai dengan yang kita inginkan
- Hal ini bisa dilakukan menggunakan function `http.ServeFile()`
- Dengan menggunakan function ini, kita bisa menentukan file mana yang ingin kita tulis ke http response

Kode : ServeFile

```
func ServeFile(writer http.ResponseWriter, request *http.Request) {
    if request.URL.Query().Get("name") != "" {
        http.ServeFile(writer, request, "./resources/ok.html")
    } else {
        http.ServeFile(writer, request, "./resources/notfound.html")
    }
}

func TestServeFile(t *testing.T) {
    server := http.Server{
        Addr:     "localhost:8080",
        Handler: http.HandlerFunc(ServeFile),
    }
}
```

Go-Lang Embed

- Parameter function `http.ServeFile` hanya berisi string file name, sehingga tidak bisa menggunakan Go-Lang Embed
- Namun bukan berarti kita tidak bisa menggunakan Go-Lang embed, karena jika untuk melakukan load file, kita hanya butuh menggunakan package `fmt` dan `ResponseWriter` saja

Kode : ServeFile Go-Lang Embed

```
//go:embed resources/ok.html
var resourceOk string

//go:embed resources/notfound.html
var resourceNotFound string

func ServeFileEmbed(writer http.ResponseWriter, request *http.Request) {
    if request.URL.Query().Get("name") != "" {
        fmt.Fprint(writer, resourceOk)
    } else {
        fmt.Fprint(writer, resourceNotFound)
    }
}
```

Template

Web Dinamis

- Sampai saat ini kita hanya membahas tentang membuat response menggunakan String dan juga static file
- Pada kenyataannya, saat kita membuat web, kita pasti akan membuat halaman yang dinamis, bisa berubah-ubah sesuai dengan data yang diakses oleh user
- Di Go-Lang terdapat fitur HTML Template, yaitu fitur template yang bisa kita gunakan untuk membuat HTML yang dinamis

HTML Template

- Fitur HTML template terdapat di package html/template
- Sebelum menggunakan HTML template, kita perlu terlebih dahulu membuat template nya
- Template bisa berubah file atau string
- Bagian dinamis pada HTML Template, adalah bagian yang menggunakan tanda {{ }}

Membuat Template

- Saat membuat template dengan string, kita perlu memberi tahu nama template nya
- Dan untuk membuat text template, cukup buat text html, dan untuk konten yang dinamis, kita bisa gunakan tanda {{.}}, contoh :
- <html><body>{{.}}</body></html>

Kode : HTML Template String

```
func SimpleHTML(writer http.ResponseWriter, request *http.Request) {  
    templateText := `<html><body>{{.}}</body></html>`  
    t, err := template.New("SIMPLE").Parse(templateText)  
    if err != nil {  
        panic(err)  
    }  
    t.ExecuteTemplate(writer, "SIMPLE", "Hello HTML Template")  
}
```



Kode : Test HTML Template String

```
request := httptest.NewRequest("GET", "http://localhost/", nil)
recorder := httptest.NewRecorder()

SimpleHTML(recorder, request)

response := recorder.Result()
body, _ := io.ReadAll(response.Body)
fmt.Println(string(body))
```

Template Dari File

- Selain membuat template dari string, kita juga bisa membuat template langsung dari file
- Hal ini mempermudah kita, karena bisa langsung membuat file html
- Saat membuat template menggunakan file, secara otomatis nama file akan menjadi nama template nya, misal jika kita punya file simple.html, maka nama template nya adalah simple.html

Kode : Simple Template



The image shows a screenshot of a code editor with a dark theme. The title bar of the window says "simple.gohtml". The code editor displays the following content:

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>{{.}}</title>
6  </head>
7  <body>
8      <h1>{{.}}</h1>
9  </body>
10 </html>
```

The code is a simple Go template. It starts with a doctype declaration, followed by an HTML tag with the language set to English. Inside the HTML tag, there is a head section containing a meta tag for UTF-8 encoding and a title tag with a placeholder {{.}}. Below the head is a body section containing a single h1 tag with the same placeholder. Finally, the entire structure is closed with an html tag.



Kode : Simple HTML Template

```
func SimpleHTMLFile(writer http.ResponseWriter, request *http.Request) {  
    t, err := template.ParseFiles("./templates/simple.gohtml")  
    if err != nil : err *  
  
    t.ExecuteTemplate(writer, "simple.gohtml", "Hello HTML Template")  
}
```

Kode : Test Simple HTML Template

```
request := httptest.NewRequest("GET", "http://localhost/", nil)
recorder := httptest.NewRecorder()

SimpleHTMLFile(recorder, request)

response := recorder.Result()
body, _ := io.ReadAll(response.Body)
fmt.Println(string(body))
```

Template Directory

- Kadang biasanya kita jarang sekali menyebutkan file template satu persatu
- Alangkah baiknya untuk template kita simpan di satu directory
- Go-Lang template mendukung proses load template dari directory
- Hal ini memudahkan kita, sehingga tidak perlu menyebutkan nama file nya satu per satu

Kode : Template Directory

```
func TemplateDirectory(writer http.ResponseWriter, request *http.Request) {  
    t, err := template.ParseGlob("./templates/*.gohtml")  
    if err != nil {  
        panic(err)  
    }  
  
    t.ExecuteTemplate(writer, "simple.gohtml", "Hello HTML Template")  
}
```

Template dari Go-Lang Embed

- Sejak Go-Lang 1.16, karena sudah ada Go-Lang Embed, jadi direkomendasikan menggunakan Go-Lang embed untuk menyimpan data template
- Menggunakan Go-Lang embed menjadi kita tidak perlu ikut meng-copy template file lagi, karena sudah otomatis di embed di dalam distribution file

Kode : Template Go-Lang Embed

```
//go:embed templates/*.gohtml
var templates embed.FS

func TemplateEmbed(writer http.ResponseWriter, request *http.Request) {
    t, err := template.ParseFS(templates, "templates/*.gohtml")
    if err != nil {
        panic(err)
    }

    t.ExecuteTemplate(writer, "simple.gohtml", "Hello HTML Template")
}
```

Template Data

Template Data

- Saat kita membuat template, kadang kita ingin menambahkan banyak data dinamis
- Hal ini bisa kita lakukan dengan cara menggunakan data struct atau map
- Namun perlu dilakukan perubahan di dalam text template nya, kita perlu memberi tahu Field atau Key mana yang akan kita gunakan untuk mengisi data dinamis di template
- Kita bisa menyebutkan dengan cara seperti ini {{.NamaField}}

Kode : Template File



```
name.gohtml
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>{{.Title}}</title>
6  </head>
7  <body>
8      <h1>Hello {{.Name}}</h1>
9  </body>
10 </html>
```

Kode : Template Data dengan Map

```
func TemplateDataStruct(writer http.ResponseWriter, request *http.Request) {  
    t := template.Must(template.ParseFiles("./templates/name.gohtml"))  
  
    t.ExecuteTemplate(writer, "name.gohtml", map[string]interface{}{
        "Title": "Template Data Struct",
        "Name":  "Eko",
    })
}
```

Kode : Template Data dengan Struct

```
func TemplateDataStruct(writer http.ResponseWriter, request *http.Request) {  
    t := template.Must(template.ParseFiles("./templates/name.gohtml"))  
  
    t.ExecuteTemplate(writer, "name.gohtml", Page{  
        Title: "Template Data Struct",  
        Name:  "Eko",  
    })  
}
```

Template Action

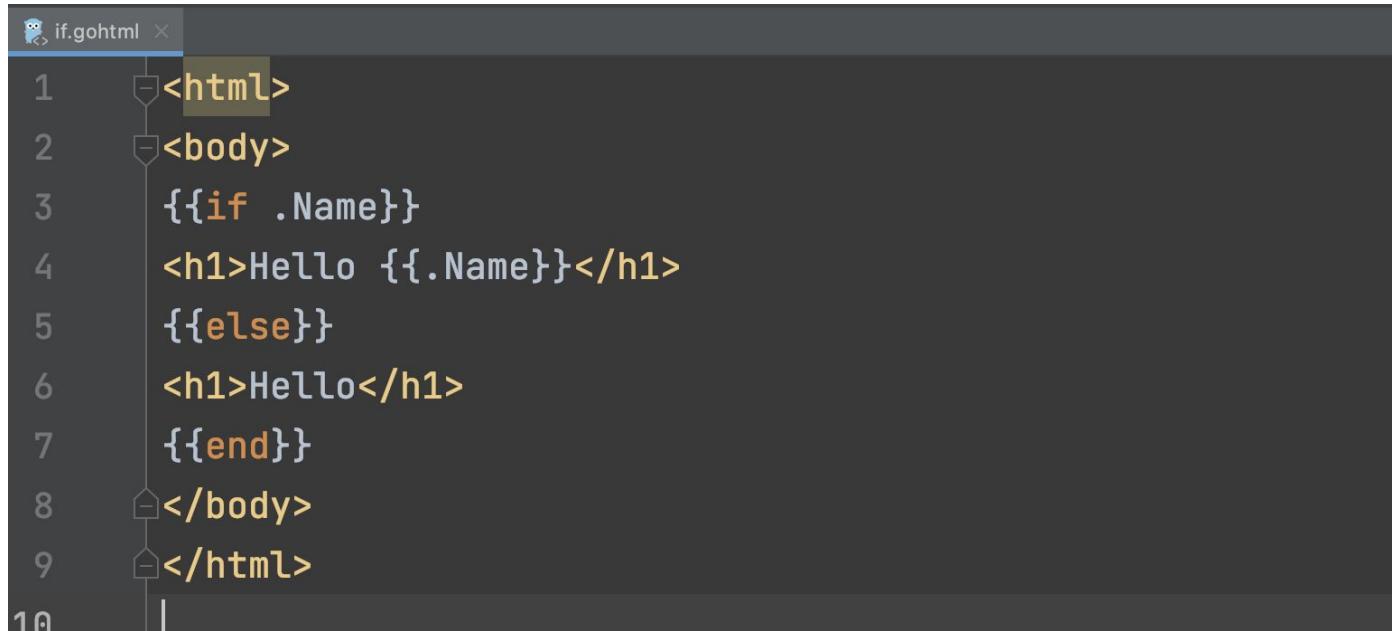
Template Action

- Go-Lang template mendukung perintah action, seperti percabangan, perulangan dan lain-lain

If Else

- {{if .Value}} T1 {{end}}, jika Value tidak kosong, maka T1 akan dieksekusi, jika kosong, tidak ada yang dieksekusi
- {{if .Value}} T1 {{else}} T2 {{end}}, jika value tidak kosong, maka T1 akan dieksekusi, jika kosong, T2 yang akan dieksekusi
- {{if .Value1}} T1 {{else if .Value2}} T2 {{else}} T3 {{end}}, jika Value1 tidak kosong, maka T1 akan dieksekusi, jika Value2 tidak kosong, maka T2 akan dieksekusi, jika tidak semuanya, maka T3 akan dieksekusi

Kode : Template If Statement



```
if.gohtml
1 <html>
2 <body>
3 {{if .Name}}
4 <h1>Hello {{.Name}}</h1>
5 {{else}}
6 <h1>Hello</h1>
7 {{end}}
8 </body>
9 </html>
```

Kode : If Statement

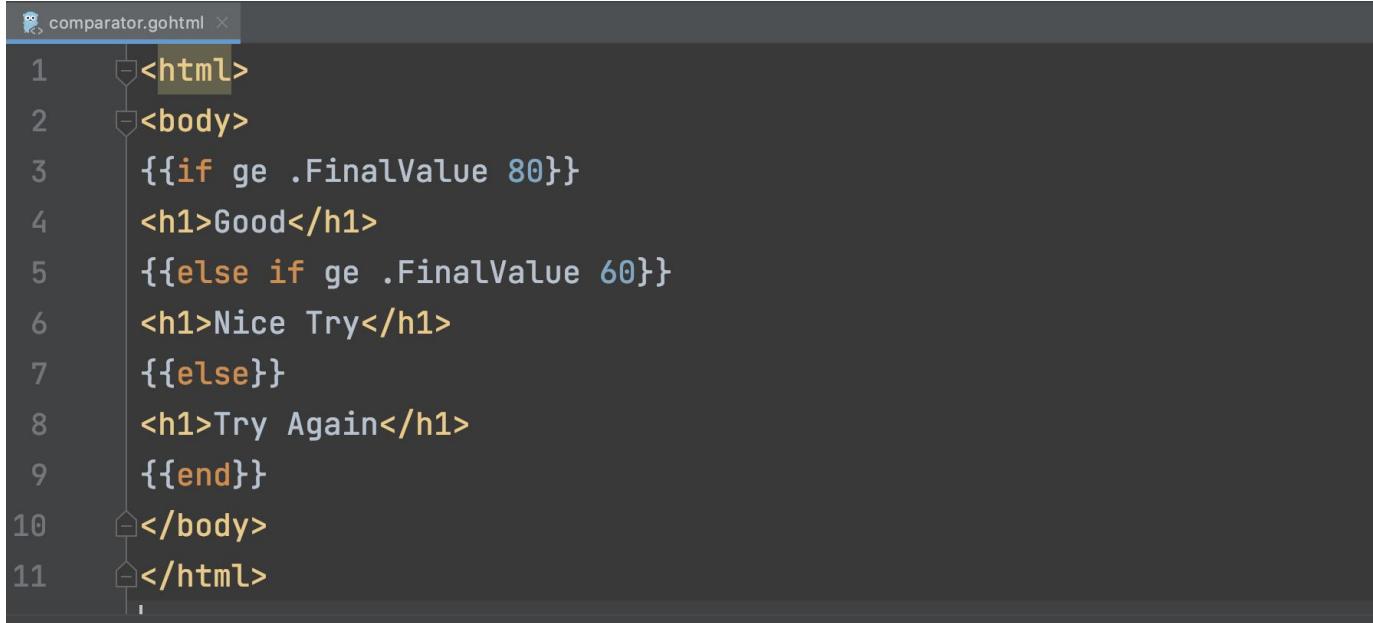
```
func TemplateIf(writer http.ResponseWriter, request *http.Request) {  
    t := template.Must(template.ParseFiles("./templates/if.gohtml"))  
  
    t.ExecuteTemplate(writer, "if.gohtml", map[string]interface{}{
        "Name": "Eko",
    })
}
```

Operator Perbandingan

Go-Lang template juga mendukung operator perbandingan, ini cocok ketika butuh melakukan perbandingan number di if statement, berikut adalah operator nya :

- eq artinya $\text{arg1} == \text{arg2}$
- ne artinya $\text{arg1} != \text{arg2}$
- lt artinya $\text{arg1} < \text{arg2}$
- le artinya $\text{arg1} \leq \text{arg2}$
- gt artinya $\text{arg1} > \text{arg2}$
- ge artinya $\text{arg1} \geq \text{arg2}$

Kode : Template Operator Perbandingan



```
comparator.gohtml
1 <html>
2   <body>
3     {{if ge .FinalValue 80}}
4       <h1>Good</h1>
5     {{else if ge .FinalValue 60}}
6       <h1>Nice Try</h1>
7     {{else}}
8       <h1>Try Again</h1>
9     {{end}}
10    </body>
11  </html>
```

Kenapa Operatornya di Depan?

- Hal ini dikarenakan, sebenarnya operator perbandingan tersebut adalah sebuah function
- Jadi saat kita menggunakan {{eq First Second}}, sebenarnya dia akan memanggil function eq dengan parameter First dan Second : eq(First, Second)

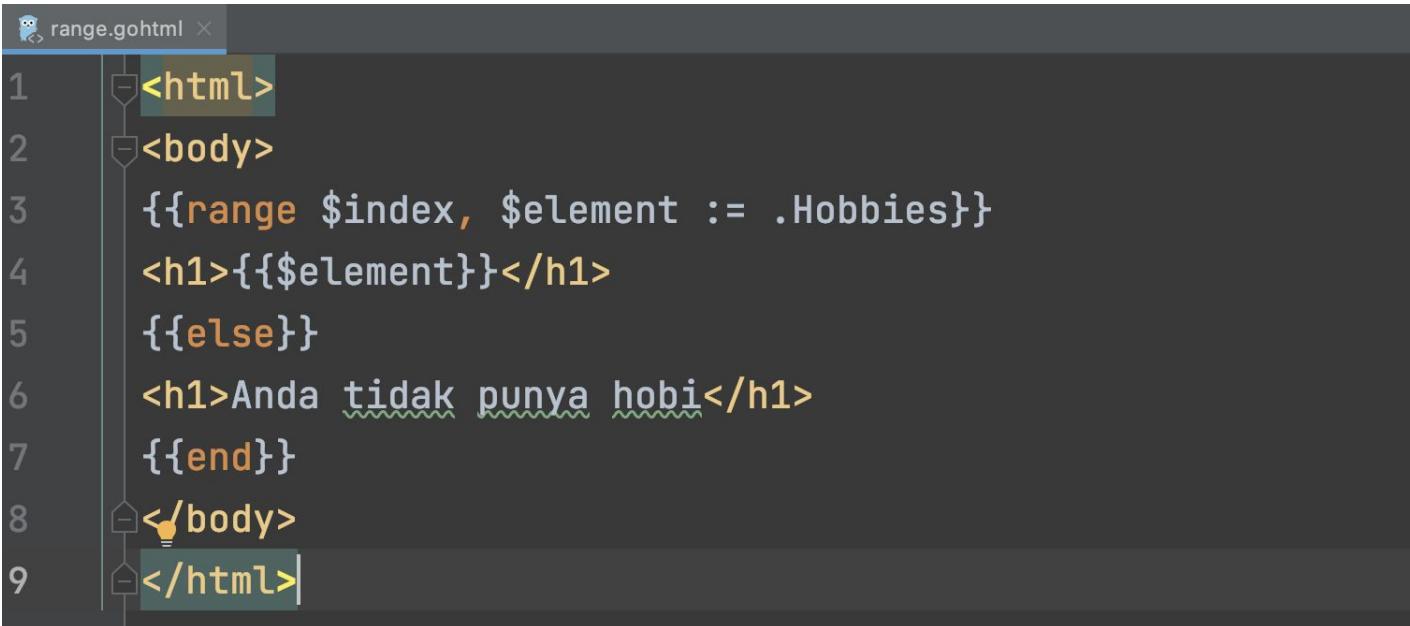
Kode : Operator Perbandingan

```
func TemplateComparator(writer http.ResponseWriter, request *http.Request) {  
    t := template.Must(template.ParseFiles("./templates/comparator.gohtml"))  
  
    t.ExecuteTemplate(writer, "comparator.gohtml", map[string]interface{}{
        "FinalValue": 100,
    })
}
```

Range

- Range digunakan untuk melakukan iterasi data template
- Tidak ada perulangan biasa seperti menggunakan for di Go-Lang template
- Yang kita bisa lakukan adalah menggunakan range untuk mengiterasi tiap data array, slice, map atau channel
- {{range \$index, \$element := .Value}} T1 {{end}}, jika value memiliki data, maka T1 akan dieksekusi sebanyak element value, dan kita bisa menggunakan \$index untuk mengakses index dan \$element untuk mengakses element
- {{range \$index, \$element := .Value}} T1 {{else}} T2 {{end}}, sama seperti sebelumnya, namun jika value tidak memiliki element apapun, maka T2 yang akan dieksekusi

Kode : Template Range



The screenshot shows a code editor window with a dark theme. The file is named "range.gohtml". The code uses a Go template to generate HTML. It starts with an opening HTML tag, followed by a body section. Inside the body, there is a range loop that iterates over the ".Hobbies" slice. If the slice is not empty, it prints an H1 tag for each element. If the slice is empty, it prints an H1 tag with the message "Anda tidak punya hobi". The template ends with a closing body tag and a closing HTML tag.

```
range.gohtml
<html>
<body>
{{range $index, $element := .Hobbies}}
<h1>$element</h1>
{{else}}
<h1>Anda tidak punya hobi</h1>
{{end}}
</body>
</html>
```

Kode : Range

```
func TemplateRange(writer http.ResponseWriter, request *http.Request) {  
    t := template.Must(template.ParseFiles("./templates/range.gohtml"))  
  
    t.ExecuteTemplate(writer, "range.gohtml", map[string]interface{}{
        "Hobbies": []string{
            "Gaming", "Reading", "Coding",
        },
    })
}
```

With

- Kadang kita sering membuat nested struct
- Jika menggunakan template, kita bisa mengaksesnya menggunakan .Value.NestedValue
- Di template terdapat action with, yang bisa digunakan mengubah scope dot menjadi object yang kita mau
- {{with .Value}} T1 {{end}}, jika value tidak kosong, di T1 semua dot akan merefer ke value
- {{with .Value}} T1 {{else}} T2 {{end}}, sama seperti sebelumnya, namun jika value kosong, maka T2 yang akan dieksekusi

Kode : Template With



```
address.gohtml
1 <html>
2 <body>
3 {{/* Contoh Komentar */}}
4 Name : {{.Name}}<br/>
5 {{with .Address}}
6 Address Street : {{.Street}}<br/>
7 Address City : {{.City}}<br/>
8 {{end}}
9 </body>
10 </html>
```

Kode : With

```
func TemplateWith(writer http.ResponseWriter, request *http.Request) {  
    t := template.Must(template.ParseFiles("./templates/address.gohtml"))  
  
    t.ExecuteTemplate(writer, "address.gohtml", map[string]interface{}{
        "Name": "Eko",
        "Address": map[string]interface{}{
            "Street": "Jalan Belum Jadi",
            "City": "Mars",
        },
    })
}
```

Comment

- Template juga mendukung komentar
- Komentar secara otomatis akan hilang ketika template text di parsing
- Untuk membuat komentar sangat sederhana, kita bisa gunakan {{/* Contoh Komentar */}}

Template Layout

Template Layout

- Saat kita membuat halaman website, kadang ada beberapa bagian yang selalu sama, misal header dan footer
- Best practice nya jika terdapat bagian yang selalu sama, disarankan untuk disimpan pada template yang terpisah, agar bisa digunakan di template lain
- Go-Lang template mendukung import dari template lain

Import Template

Untuk melakukan import, kita bisa menggunakan perintah berikut :

- {{template "nama"}}, artinya kita akan meng-import template "nama" tanpa memberikan data apapun
- {{template "nama" .Value}}, artinya kita akan meng-import template "nama" dengan memberikan data value

Kode : File Template Header

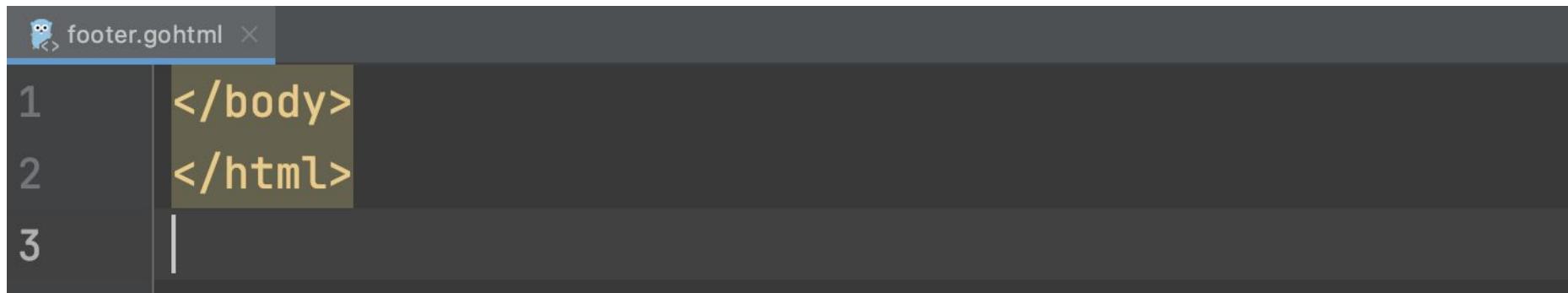


The screenshot shows a code editor window with a dark theme. The title bar of the window says "header.gohtml". The code itself is an HTML template:

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>{{.Title}}</title>
6  </head>
7  <body>
```

The code is numbered from 1 to 7 on the left. Lines 1 through 5 are indented under the opening `<head>` tag, and lines 6 and 7 are indented under the closing `</head>` tag. The `<body>` tag at line 7 is currently selected, as indicated by a dark teal background.

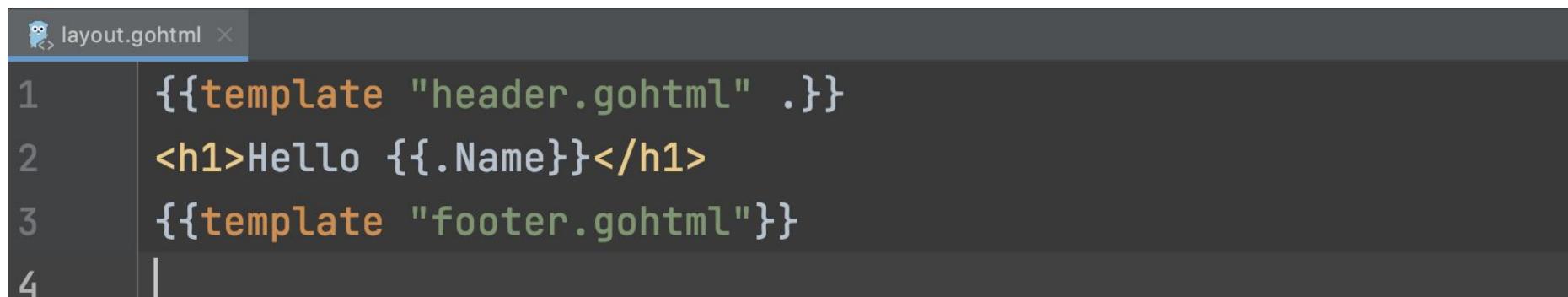
Kode : File Template Footer



A screenshot of a code editor showing a file named "footer.gohtml". The file contains three lines of code: 1. </body>, 2. </html>, and 3. An empty line. The code editor has a dark theme with syntax highlighting. The file tab shows a small owl icon.

```
1 </body>
2 </html>
3 |
```

Kode : File Template



The screenshot shows a code editor window with a dark theme. The title bar of the window displays a blue owl icon, the file name "layout.gohtml", and a close button. The main area of the editor contains four lines of Go template code:

```
1 {{template "header.gohtml" .}}
2 <h1>Hello {{.Name}}</h1>
3 {{template "footer.gohtml"}}
4 |
```



Kode : Template Layout

```
func TemplateLayout(writer http.ResponseWriter, request *http.Request) {
    t := template.Must(template.ParseFiles(
        "./templates/header.gohtml", "./templates/footer.gohtml", "./templates/layout.gohtml",
    ))
    t.ExecuteTemplate(writer, "layout.gohtml", map[string]interface{}{
        "Name": "Eko",
        "Title": "Template Layout",
    })
}
```

Template Name

- Saat kita membuat template dari file, secara otomatis nama file nya akan menjadi nama template
- Namun jika kita ingin mengubah nama template nya, kita juga bisa melakukan menggunakan perintah {{define "nama"}} TEMPLATE {{end}}, artinya kita membuat template dengan nama “nama”

Kode : Template Name

```
layout.gohtml ×  
1 {{define "layout"}}  
2 {{template "header.gohtml" .}}  
3 <h1>Hello {{.Name}}</h1>  
4 {{template "footer.gohtml"}}  
5 {{end}}  
6
```

Template Function

Template Function

- Selain mengakses field, dalam template, function juga bisa diakses
- Cara mengakses function sama seperti mengakses field, namun jika function tersebut memiliki parameter, kita bisa gunakan tambahkan parameter ketika memanggil function di template nya
- {{.FunctionName}}, memanggil field FunctionName atau function FunctionName()
- {{.FunctionName "eko", "kurniawan"}}, memanggil function FunctionName("eko", "kurniawan")

Kode : Struct

```
type MyPage struct {  
    Name string  
}  
  
func (myPage MyPage) SayHello(name string) string {  
    return "Hello " + name + ", My Name is " + myPage.Name  
}
```

Kode : Template Function

```
func TemplateFunction(writer http.ResponseWriter, request *http.Request) {  
    t := template.Must(template.New("FUNCTION").  
        Parse(`{{ .SayHello "Budi" }}`))  
  
    t.ExecuteTemplate(writer, "FUNCTION", MyPage{  
        Name: "Eko",  
    })  
}
```

Global Function

- Go-Lang template memiliki beberapa global function
- Global function adalah function yang bisa digunakan secara langsung, tanpa menggunakan template data
- Berikut adalah beberapa global function di Go-Lang template
- <https://github.com/golang/go/blob/master/src/text/template/funcs.go>

Kode : Global Fuction

```
func TemplateFunctionGlobal(writer http.ResponseWriter, request *http.Request) {  
    t := template.Must(template.New("FUNCTION").  
        Parse(`{{len .Name}}`))  
  
    t.ExecuteTemplate(writer, "FUNCTION", map[string]interface{}{
        "Name": "Tutorial Go-Lang",
    })
}
```

Menambah Global Function

- Kita juga bisa menambah global function
- Untuk menambah global function, kita bisa menggunakan method `Funcs` pada template
- Perlu diingat, bahwa menambahkan global function harus dilakukan sebelum melakukan parsing template

Kode : Menambah Global Function

```
func TemplateFunctionMap(writer http.ResponseWriter, request *http.Request) {  
    t := template.New("FUNCTION")  
    t = t.Funcs(map[string]interface{}``{  
        "upper": func(value string) string {  
            return strings.ToUpper(value)  
        },  
    })  
    t = template.Must(t.Parse(`{{ upper .Name }}`))  
  
    t.ExecuteTemplate(writer, "FUNCTION", MyPage{  
        Name: "Eko",  
    })  
}
```

Function Pipelines

- Go-Lang template mendukung function pipelines, artinya hasil dari function bisa dikirim ke function berikutnya
- Untuk menggunakan function pipelines, kita bisa menggunakan tanda | , misal :
- {{ sayHello .Name | upper }}, artinya akan memanggil global function sayHello(Name) hasil dari sayHello(Name) akan dikirim ke function upper(hasil)
- Kita bisa menambahkan function pipelines lebih dari satu

Kode : Function Pipelines

```
func TemplateFunctionPipelines(writer http.ResponseWriter, request *http.Request) {
    t := template.New("FUNCTION")
    t = t.Funcs(map[string]interface{}``{
        "sayHello": func(value string) string {
            return "Hello " + value
        },
        "upper": func(value string) string {
            return strings.ToUpper(value)
        },
    })
    t = template.Must(t.Parse(`{{ sayHello .Name | upper }}`))

    t.ExecuteTemplate(writer, "FUNCTION", MyPage{
        Name: "Eko",
    })
}
```

Template Caching

Template Caching

- Kode-kode diatas yang sudah kita praktekan sebenarnya tidak efisien
- Hal ini dikarenakan, setiap Handler dipanggil, kita selalu melakukan parsing ulang template nya
- Idealnya template hanya melakukan parsing satu kali diawal ketika aplikasinya berjalan
- Selanjutnya data template akan di caching (disimpan di memory), sehingga kita tidak perlu melakukan parsing lagi
- Hal ini akan membuat web kita semakin cepat



Kode : Template Caching

```
//go:embed templates/*.gohtml
var templates embed.FS

var myTemplates = template.Must(template.ParseFS(templates, "templates/*.gohtml"))

func TemplateCaching(writer http.ResponseWriter, request *http.Request) {
    myTemplates.ExecuteTemplate(writer, "simple.gohtml", "Hello HTML Template")
}
```

XSS (Cross Site Scripting)

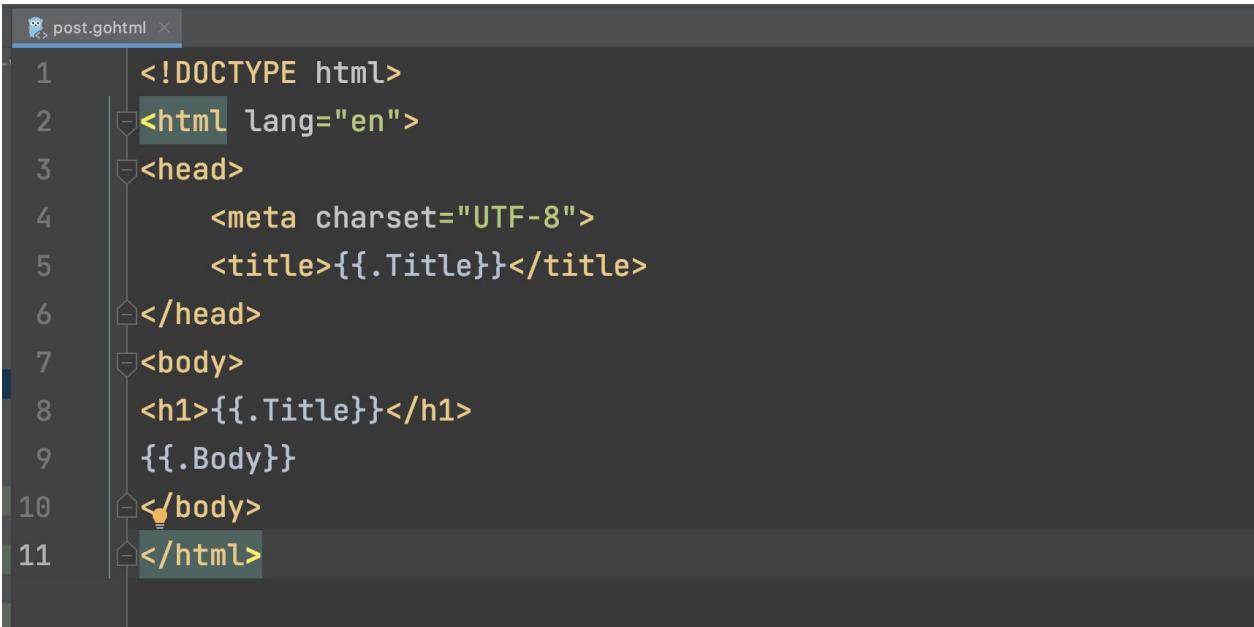
XSS (Cross Site Scripting)

- XSS adalah salah satu security issue yang biasa terjadi ketika membuat web
- XSS adalah celah keamanan, dimana orang bisa secara sengaja memasukkan parameter yang mengandung JavaScript agar dirender oleh halaman website kita
- Biasanya tujuan dari XSS adalah mencuri cookie browser pengguna yang sedang mengakses website kita
- XSS bisa menyebabkan account pengguna kita diambil alih jika tidak ditangani dengan baik

Auto Escape

- Berbeda dengan bahasa pemrograman lain seperti PHP, pada Go-Lang template, masalah XSS sudah diatasi secara otomatis
- Go-Lang template memiliki fitur Auto Escape, dimana dia bisa mendeteksi data yang perlu ditampilkan di template, jika mengandung tag-tag html atau script, secara otomatis akan di escape
- Semua function escape bisa diliat disini :
- <https://github.com/golang/go/blob/master/src/html/template/escape.go>
- <https://golang.org/pkg/html/template/#hdr-Contexts>

Kode : File Template



A screenshot of a code editor showing a file named "post.gohtml". The code is a Go template for generating HTML. It includes a DOCTYPE declaration, an HTML tag with a lang attribute set to "en", a head section containing a meta tag for UTF-8 charset and a title block that outputs the value of the .Title variable, a body section containing an H1 tag that also outputs the .Title variable and a .Body block, and finally an HTML tag.

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>{{.Title}}</title>
6  </head>
7  <body>
8      <h1>{{.Title}}</h1>
9      {{.Body}}
10     </body>
11 </html>
```

Kode : Auto Escape

```
func TemplateAutoEscape(writer http.ResponseWriter, request *http.Request) {  
    myTemplates.ExecuteTemplate(writer, "post.gohtml", map[string]interface{}{  
        "Title": "Go-Lang Auto Escape",  
        "Body": "<p>Selamat Belajar Go-Lang Web</p>",  
    })  
}
```

Mematikan Auto Escape

- Jika kita mau, auto escape juga bisa kita matikan
- Namun, kita perlu memberi tahu template secara eksplisit ketika kita menambahkan template data
- Kita bisa menggunakan data
- template.HTML , jika ini adalah data html
- template.CSS, jika ini adalah data css
- template.JS, jika ini adalah data javascript

Kode : Mematikan Auto Escape

```
func TemplateAutoEscapeDisabled(writer http.ResponseWriter, request *http.Request) {  
    myTemplates.ExecuteTemplate(writer, "post.gohtml", map[string]interface{}{
        "Title": "Go-Lang Auto Escape",
        "Body":  template.HTML("<p>Selamat Belajar Go-Lang Web</p>"),
    })
}
```

Masalah XSS (Cross Site Scripting)

- Saat kita mematikan fitur auto escape, bisa dipastikan masalah XSS akan mengintai kita
- Jadi pastikan kita benar-benar percaya terhadap sumber data yang kita matikan auto escape nya

Kode : Contoh XSS

```
func TemplateXSS(writer http.ResponseWriter, request *http.Request) {  
    myTemplates.ExecuteTemplate(writer, "post.gohtml", map[string]interface{}{
        "Title": "Go-Lang Auto Escape",
        "Body":  template.HTML(request.URL.Query().Get("body")),
    })
}
```

Redirect

Redirect

- Saat kita membuat website, kadang kita butuh melakukan redirect
- Misal setelah selesai login, kita lakukan redirect ke halaman dashboard
- Redirect sendiri sebenarnya sudah standard di HTTP
<https://developer.mozilla.org/en-US/docs/Web/HTTP/Redirections>
- Kita hanya perlu membuat response code 3xx dan menambah header Location
- Namun untungnya di Go-Lang, ada function yang bisa kita gunakan untuk mempermudah ini

Kode : Redirect

```
func RedirectTo(writer http.ResponseWriter, request *http.Request) {
    fmt.Fprint(writer, "Hello Redirect")
}

func RedirectFrom(writer http.ResponseWriter, request *http.Request) {
    http.Redirect(writer, request, "/redirect-to", http.StatusTemporaryRedirect)
}
```



Kode : Mencoba Redirect

```
mux := http.NewServeMux()
mux.HandleFunc("/redirect-from", RedirectFrom)
mux.HandleFunc("/redirect-to", RedirectTo)

server := http.Server{
    Addr:      "localhost:8080",
    Handler:   mux,
}
err := server.ListenAndServe()
```

Upload File

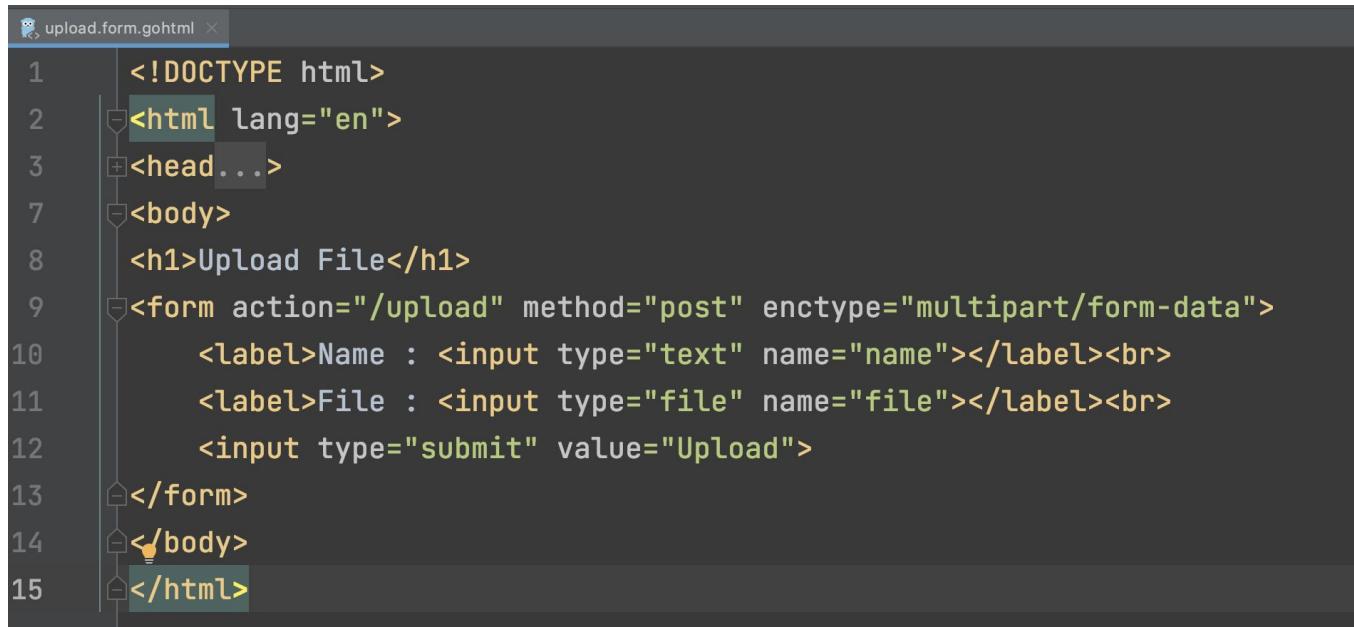
Upload File

- Saat membuat web, selain menerima input data berupa form dan query param, kadang kita juga menerima input data berupa file dari client
- Go-Lang Web sudah memiliki fitur untuk management upload file
- Hal ini memudahkan kita ketika butuh membuat web yang menerima input file upload

MultiPart

- Saat kita ingin menerima upload file, kita perlu melakukan parsing terlebih dahulu menggunakan Request.ParseMultipartForm(size), atau kita bisa langsung ambil data file nya menggunakan Request.FormFile(name), di dalam nya secara otomatis melakukan parsing terlebih dahulu
- Hasilnya merupakan data-data yang terdapat pada package multipart, seperti multipart.File sebagai representasi file nya, dan multipart.FileHeader sebagai informasi file nya

Kode : File Template Form Upload



```
upload.form.gohtml
1  <!DOCTYPE html>
2  <html lang="en">
3  <head...>
7  <body>
8  <h1>Upload File</h1>
9  <form action="/upload" method="post" enctype="multipart/form-data">
10 <label>Name : <input type="text" name="name"></label><br>
11 <label>File : <input type="file" name="file"></label><br>
12 <input type="submit" value="Upload">
13 </form>
14 </body>
15 </html>
```

Kode : Display Form Upload

```
func UploadForm(writer http.ResponseWriter, request *http.Request) {  
    err := myTemplates.ExecuteTemplate(writer, "upload.form.gohtml", nil)  
    if err != nil {  
        panic(err)  
    }  
}
```

Kode : Upload Handler

```
func Upload(writer http.ResponseWriter, request *http.Request) {  
    file, fileHeader, err := request.FormFile("file")  
    if err != nil : err *  
    fileDestination, err := os.Create("./resources/" + fileHeader.Filename)  
    if err != nil : err *  
    _, err = io.Copy(fileDestination, file)  
    if err != nil : err *  
    name := request.PostFormValue("name")  
    myTemplates.ExecuteTemplate(writer, "upload.success.gohtml", map[string]interface{}{  
        "Name": name,  
        "File": "/static/" + fileHeader.Filename,  
    })  
}
```

Kode : Template Upload Handler

```
upload.success.gohtml ×
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>Success</title>
6  </head>
7  <body>
8      <h1>{{.Name}}</h1>
9      <a href="{{.File}}>File</a>
10 </body>
11 </html>
```



Kode : Server Upload

```
mux := http.NewServeMux()
mux.HandleFunc("/", UploadForm)
mux.HandleFunc("/upload", Upload)
mux.Handle("/static/", http.StripPrefix("/static", http.FileServer(http.Dir("./resources"))))

server := http.Server{
    Addr:     "localhost:8080",
    Handler: mux,
}

err := server.ListenAndServe()
```



Kode : Test Upload

```
body := new(bytes.Buffer)
writer := multipart.NewWriter(body)
writer.WriteField("name", "Eko Kurniawan Khanedy")

file, _ := writer.CreateFormFile("file", "NewLogo.png")
file.Write(logo)
writer.Close()

request := httptest.NewRequest("POST", "http://localhost/", body)
request.Header.Set("Content-Type", writer.FormDataContentType())
recorder := httptest.NewRecorder()

Upload(recorder, request)
```

Download File

Download File

- Selain upload file, kadang kita ingin membuat halaman website yang digunakan untuk download sesuatu
- Sebenarnya di Go-Lang sudah disediakan menggunakan FileServer dan ServeFile
- Dan jika kita ingin memaksa file di download (tanpa di render oleh browser, kita bisa menggunakan header Content-Disposition)
- <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Disposition>



Kode : Download File

```
func DownloadFile(writer http.ResponseWriter, request *http.Request) {
    fileName := request.URL.Query().Get("file")
    if fileName == "" {
        writer.WriteHeader(http.StatusBadRequest)
        fmt.Fprint(writer, "BAD REQUEST")
        return
    }

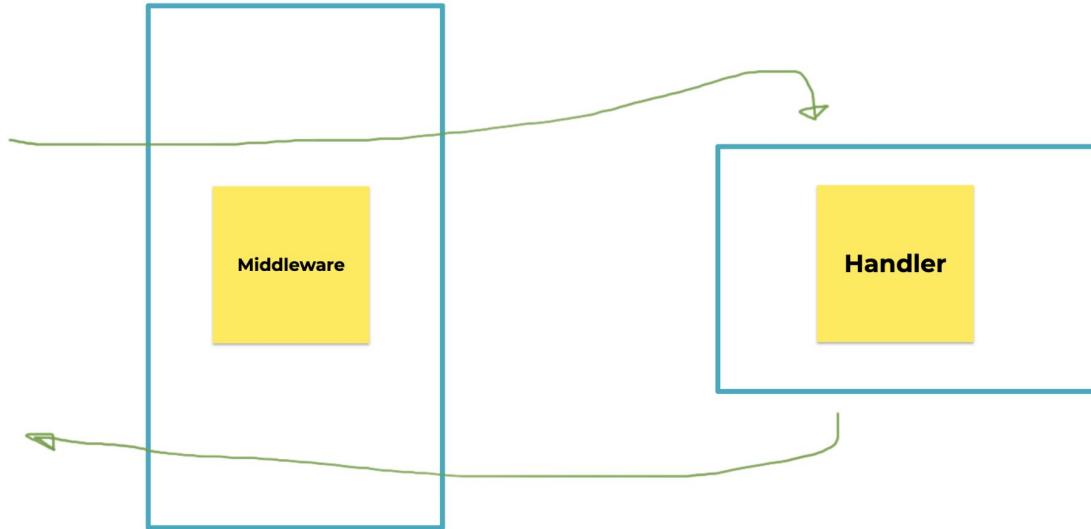
    writer.Header().Add("Content-Disposition", "attachment; filename=\"" +fileName+ "\"")
    http.ServeFile(writer, request, "./resources/" +fileName)
}
```

Middleware

Middleware

- Dalam pembuatan web, ada konsep yang bernama middleware atau filter atau interceptor
- Middleware adalah sebuah fitur dimana kita bisa menambahkan kode sebelum dan setelah sebuah handler di eksekusi

Diagram Middleware



Middleware di Go-Lang web

- Sayangnya, di Go-Lang web tidak ada middleware
- Namun karena struktur handler yang baik menggunakan interface, kita bisa membuat middleware sendiri menggunakan handler



Kode : Log Middleware

```
type LogMiddleware struct {
    Handler http.Handler
}

func (middleware *LogMiddleware) ServeHTTP(writer http.ResponseWriter, request *http.Request) {
    fmt.Println("Before Execute Handler")
    middleware.Handler.ServeHTTP(writer, request)
    fmt.Println("After Execute Handler")
}
```



Kode : Server Log Middleware

```
mux := http.NewServeMux()
mux.HandleFunc("/", func(writer http.ResponseWriter, request *http.Request) {
    fmt.Fprintf(writer, "Hello Middleware")
})
logMiddleware := new(LogMiddleware)
logMiddleware.Handler = mux

server := http.Server{
    Addr:      "localhost:8080",
    Handler:  logMiddleware,
}
err := server.ListenAndServe()
```

Error Handler

- Kadang middleware juga bisa digunakan untuk melakukan error handler
- Hal ini sehingga jika terjadi panic di Handler, kita bisa melakukan recover di middleware, dan mengubah panic tersebut menjadi error response
- Dengan ini, kita bisa menjaga aplikasi kita tidak berhenti berjalan

Kode : Error Handler Middleware

```
type ErrorHandler struct {
    Handler http.Handler
}

func (handler *ErrorHandler) ServeHTTP(writer http.ResponseWriter, request *http.Request) {
    defer func() {
        err := recover()
        fmt.Println("RECOVER : ", err)
        if err != nil {
            writer.WriteHeader(http.StatusInternalServerError)
            fmt.Fprintf(writer, "Error : %s", err)
        }
    }()
    handler.Handler.ServeHTTP(writer, request)
}
```



Kode : Server Error Handler Middleware

```
mux := http.NewServeMux()
mux.HandleFunc("/", func(writer http.ResponseWriter, request *http.Request) {
    panic("Ups, Error Happens")
})
errorHandler := &ErrorHandler{Handler: &LogMiddleware{Handler: mux}}

server := http.Server{
    Addr:      "localhost:8080",
    Handler:  errorHandler,
}
err := server.ListenAndServe()
```

Routing Library

Routing Library

- Walaupun Go-Lang sudah menyediakan ServeMux sebagai handler yang bisa menghandle beberapa endpoint atau istilahnya adalah routing
- Tapi kebanyakan programmer Go-Lang biasanya akan menggunakan library untuk melakukan routing
- Hal ini dikarenakan ServeMux tidak memiliki advanced fitur seperti path variable, auto binding parameter dan middleware
- Banyak alternatif lain yang bisa kita gunakan untuk library routing selain ServeMux

Contoh Routing Library

- <https://github.com/julienschmidt/httprouter>
- <https://github.com/gorilla/mux>
- Dan masih banyak lagi : <https://github.com/julienschmidt/go-http-routing-benchmark>

Tutorial Routing Library

- Tutorial routing library tidak akan dibahas di course ini
- Course ini hanya fokus ke standard http package Go-Lang
- Routing Library akan dibahas di course terpisah

Materi Selanjutnya

Materi Selanjutnya

- Go-Lang RESTful API
- Go-Lang Deployment
- Go-Lang Docker