Scalable Web Service With Golang sesi 6

Web Server

Web Server #1

Go telah menyediakan sebuah package bernama *net/http* untuk berbagai macam keperluan dalam membuat sebuah aplikasi berbasis web seperti contohnya routing, templating, web server dan lain-lain. Dan pada materi kali ini kita akan belajar bagaimana cara membuat web server beserta routingnya pada bahasa Go.

Sekarang buatlah satu file dengan nama **web.go** dan isilah dengan syntax seperti pada gambar di sebelah kanan.

```go
package main

import (
  "fmt"
  "net/http"
)

var PORT = ":8080"

func main() {
  http.HandleFunc("/", greet)

  http.ListenAndServe(PORT, nil)
}

func greet(w http.ResponseWriter, r *http.Request) {
  msg := "Hello World"
  fmt.Fprint(w, msg)
}
```

Web Server #2

Pada line 8, kita mempunyai sebuah variable global bernama *PORT* yang dimana kita menyimpan nilai port locahost yang mengarah pada localhost:8080.

Lalu pada line 11, kita memakai function *HandleFunc* yang berasal dari package *http*. Function *HandleFunc* digunakan untuk keperluan routing kita, yang dimana function tersebut menerima 2 parameter.

Parameter pertama digunakan untuk mendefinisikan path routingnya, sedangkan parameter kedua menerima sebuah function dengan 2 parameter yaitu http.ResponseWriter dan pointer http.Request

```
1  package main
2
3  import (
4    "fmt"
5    "net/http"
6  )
7
8  var PORT = ":8080"
9
10 func main() {
11   http.HandleFunc("/", greet)
12
13   http.ListenAndServe(PORT, nil)
14 }
15
16 func greet(w http.ResponseWriter, r *http.Request) {
17   msg := "Hello World"
18   fmt.Fprint(w, msg)
19 }
```

Web Server #3

http.ResponseWriter adalah sebuah interface dengan berbagai method yang digunakan untuk mengirim response balik kepada client yang mengirimkan request. Kemudian *http.Request* adalah sebuah struct yang digunakan untuk mendapat berbagai macam data request seperti form value, headers, url parameter dan lain-lain.

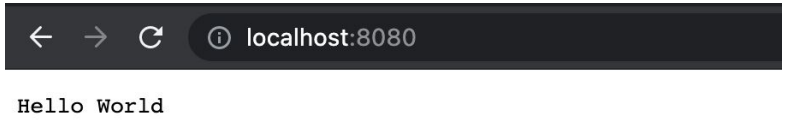
Kemudian pada line 13, kita menggunakan function *ListenAndServe* untuk menjalankan server aplikasi. Function *ListenAndServe* menerima 2 parameter yaitu keterangan port yang kita pakai, dan http.Handler yang merupakan sebuah *interface*. Namun karena kita tidak menggunakan *http.Handler*, maka kita cukup memberikan zero value dari tipe data *interface* yaitu *nil*.

Kemudian pada line 16 - 19, kita membuat function *greet* yang dimana kita gunakan untuk mengirim response berupa tulisan "Hello World" kepada client. Lalu untuk mengirim response nya kita menggunakan function *fmt.Fprint*.

```
1  package main
2
3  import (
4    "fmt"
5    "net/http"
6  )
7
8  var PORT = ":8080"
9
10 func main() {
11   http.HandleFunc("/", greet)
12
13   http.ListenAndServe(PORT, nil)
14 }
15
16 func greet(w http.ResponseWriter, r *http.Request) {
17   msg := "Hello World"
18   fmt.Fprint(w, msg)
19 }
```

Web Server #4

Sekarang jalankan server aplikasi kita pada terminal dengan perintah **go run web.go**. Kemudian bukalah browser dan arahkan urk kepada http://localhost:8080/. Dan kita akan melihat sebuah tulisan berupa **Hello World** seperti pada gambar di bawah.



API #1

Sekarang kita akan mencoba untuk membuat sebuah API yang sangat simple. Yang dimana kita akan menggunakan 2 method saja berupa GET dan POST yang dimana akan kita gunakan untuk mendapatkan data employee dan membuat data employee baru.

Untuk itu kita akan membuat data-data employee itu terlebih dahulu dengan menyimpannya didalam sebuah variable dengan tipe data *slice* yang berisikan tipe data struct *Employee*.

Maka dari itu, buat lah sebuah file dengan nama api.go, lalu kemudian ikutilah syntax-syntax seperti pada gambar disebelah kanan.

```go
package main

import (
  "encoding/json"
  "fmt"
  "net/http"
  "strconv"
)

type Employee struct {
  ID       int
  Name     string
  Age      int
  Division string
}

var employees = []Employee{
  {ID: 1, Name: "Airell", Age: 23, Division: "IT"},
  {ID: 2, Name: "Nanda", Age: 23, Division: "Finance"},
  {ID: 3, Name: "Mailo", Age: 20, Division: "IT"},
}

var PORT = ":8080"

```

API #2

Kemudian ikutilah syntax seperti pada gambar disebelah kanan untuk function *main*, dan juga untuk function *getEmployees*.

Pada line 33, kita menggunakan method *Header* dari interface *http.ResponseWriter* yang kemudian di chaining dengan method *Set*.

Hal ini kita lakukan untuk menentukan bentuk dari data response yang ingin kita kirimkan kepada client. Karena saat ini kita ingin mengirim response data dalam bentuk JSON, maka kita dapat mengatur *Content-Type* nya menjadi *application/json* dalam method *Set*.

```
25  func main() {
26    http.HandleFunc("/employees", getEmployees)
27
28    fmt.Println("Application is listening on port", PORT)
29    http.ListenAndServe(PORT, nil)
30  }
31
32  func getEmployees(w http.ResponseWriter, r *http.Request) {
33    w.Header().Set("Content-Type", "application/json")
34
35    if r.Method == "GET" {
36      json.NewEncoder(w).Encode(employees)
37      return
38    }
39
40    http.Error(w, "Invalid method", http.StatusBadRequest)
41  }
```

API #3

Lalu pada line 35 kita melakukan pengecekan *method*. Karena function *getEmployees* kita gunakan untuk mendapatkan data-data empoyee dan pada umumnya untuk mendapatkan data dari server menggunakan method *GET*.

Lalu pada line 36, kita mengkonversi data *employees* menjadi data berbentuk JSON untuk dikirimkan kepada client dengan menggunakan method *NewEncoder* yang berasal dari package *json*, yang kemudian kita chaining dengan method *Encode* untuk mengkonversi datanya menjadi bentuk JSON.

```
25  func main() {
26    http.HandleFunc("/employees", getEmployees)
27
28    fmt.Println("Application is listening on port", PORT)
29    http.ListenAndServe(PORT, nil)
30  }
31
32  func getEmployees(w http.ResponseWriter, r *http.Request) {
33    w.Header().Set("Content-Type", "application/json")
34
35    if r.Method == "GET" {
36      json.NewEncoder(w).Encode(employees)
37      return
38    }
39
40    http.Error(w, "Invalid method", http.StatusBadRequest)
41  }
```

API #4

Kemudian jika method yang dikirimkan oleh client bukan method *GET,* maka kita akan mengirimkan response error dengan menggunakan function Error dari package *http*.

Lalu *http.StatusBadRequest* merupakan sebuah konstanta dari package *http.StatusBadRequest* yang merepresentasikan status *400*.

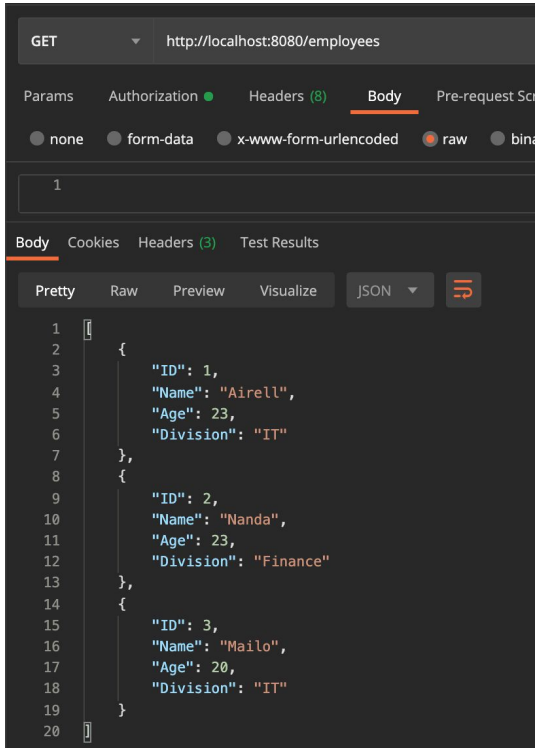
```
25  func main() {
26    http.HandleFunc("/employees", getEmployees)
27
28    fmt.Println("Application is listening on port", PORT)
29    http.ListenAndServe(PORT, nil)
30  }
31
32  func getEmployees(w http.ResponseWriter, r *http.Request) {
33    w.Header().Set("Content-Type", "application/json")
34
35    if r.Method == "GET" {
36      json.NewEncoder(w).Encode(employees)
37      return
38    }
39
40    http.Error(w, "Invalid method", http.StatusBadRequest)
41  }
```

API #5

Sekarang kita akan mencoba untuk mengirimkan request kepada server kita dengan menggunakan *Postman*.

Jalankan server aplikasi kita dengan perintah **go run api.go**, dan buat request pada postman yang mengarah pada URL http://localhost:8080/employees dengan method GET.

Maka setelah itu kita akan mendapatkan data response berbentuk JSON seperti pada gambar di sebelah kanan.



API #6

Sebelumnya kita telah menggunakan method *GET* untuk mendapatkan seluruh data employee. Unutk sekarang kita akan mencoba menerakan method *POST* yang dimana kita gunakan untuk menambah data employee baru.

Maka dari itu buatlah sebuah function baru dengan nama createEmployee seperti pada gambar di sebelah kanan.

Pada line 49 - 51, kita mencoba untuk mendapatkan nilai input form dari client dengan menggunakan method *FormValue* yang berasal dari struct *http.Request*.

Kemudian pada line 53 kita mengkonversi input *age* dari tipe data string menjadi tipe data *int* karena seluruh nilai input yang kita dapat dari client akan memiliki tipe data *string*, sedangkan field *age* pada struct *Employee* memiliki tipe data *int*. Setelah itu pada line 60, kita membuat variable *newEmployee* yang akan menampung nilai-nilai input dari client yang kemudian akan kita append atau masukkan kedalam slice *employees*.

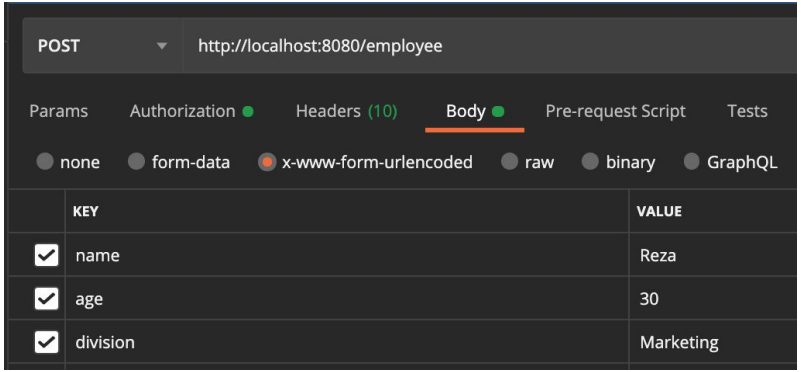
```
45  func createEmployee(w http.ResponseWriter, r *http.Request) {
46    w.Header().Set("Content-Type", "application/json")
47
48    if r.Method == "POST" {
49      name := r.FormValue("name")
50      age := r.FormValue("age")
51      divison := r.FormValue("division")
52
53      convertAge, err := strconv.Atoi(age)
54
55      if err != nil {
56        http.Error(w, "Invalid age", http.StatusBadRequest)
57        return
58      }
59
60      newEmployee := Employee{
61        ID:       len(employees) + 1,
62        Name:     name,
63        Age:      convertAge,
64        Division: divison,
65      }
66
67      employees = append(employees, newEmployee)
68
69      json.NewEncoder(w).Encode(newEmployee)
70      return
71    }
72
73    http.Error(w, "Invalid method", http.StatusBadRequest)
74  }
```

API #7

Sebelumnya kita mencoba untuk mengirimkan request menggunakan Postman, kita perlu meregistrasikan function *createEmployee* kedalam routingan kita.

Kemudian jalankan ulang server aplikasi kita, lalu buatlah nilai-nilai input pada Postman seperti pada gambar kedua.

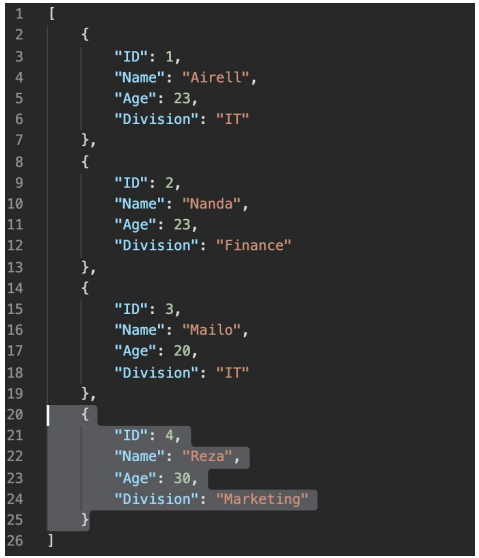
```go
func main() {
  http.HandleFunc("/employees", getEmployees)

  http.HandleFunc("/employee", createEmployee)

  fmt.Println("Application is listening on port", PORT)
  http.ListenAndServe(PORT, nil)
}
```



API #8

Sekarang kirimkan request dari Postman yang mengarah pada URL http://localhost:8080/employee dengan method *POST*. Maka kita akan mendapatkan data response seperti pada gambar pertama disebelah kanan.

Kemudian jika kita kirimkan request kembali yang mengarah pada URL http://localhost:8080/employees dengan method *GET*, maka kita akan dapat melihat data employee yang baru saja kita buat sudah ada di dalam kumpulan data employee nya seperti pada gambar kedua.



API #9

Kita juga dapat mengirimkan data employee dengan dengan template html. Bahasa Go telah menyediakan suatu package bernama *html/template* untuk melakukannya.

Sekarang buatlah satu file html dengan nama **template.html** yang masih berada pada direktori yang sama dengan file **api.go** yang kita buat sebelumnya. Kemudian ikutilah syntax seperti pada gambar di sebelah kanan untuk file htmlnya.

Jika kita perhatikan, terdapat tanda seperti {{.}}.

Ketika kita ingin menampilkan data pada file html, maka kita perlu menyelipkan data tersebut kedalam dua tanda kurung {{}}, lalu menempatkan tand titik didalamnya seperti yang kita lihat pada gambar di sebelah kanan.

Tanda titik . yang berada di dalam 2 tanda kurung tersebut merepresentasikan data yang ingin kita tampilkan.

```
<html>
  <head>
    <title>HTML</title>
  </head>
  <body>
    <div>
      {{.}}
    </div>
    </body>
</html>
```

API #10

Kemudian kita akan mengubah sedikit isi dari function *getEmployees* yang sudah kita buat sebelumnya menjadi seperti pada gambar di sebelah kanan.

Pada line 38, kita menggunakan function *template.ParseFiles* yang berasal dari package *html/template* yang digunakan untuk mem-parsing file html kita. Karena file html yang baru kita buat bernama **template.html**, maka dari itu kita meletakkan nama file html kita sebagai argumen function *template.ParseFiles*.

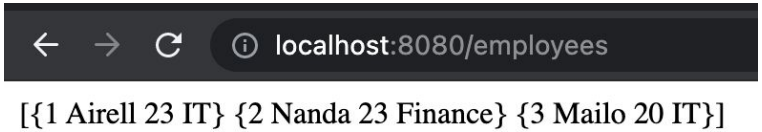
Kemudian jika sudah berhasil di parsing oleh function *template.ParseFiles*, maka function tersebut akan mengembaikan suatu tipe data struct *template*.

Template yang mempunyai sebuah method bernama *Execute* yang digunakan untuk memberikan response kepada client berupa template html. Parameter kedua dari method *Execute* digunakan untuk mengirimkan data yang ingin kita tampilkan pada file html kita.

```
35  func getEmployees(w http.ResponseWriter, r *http.Request) {
36
37    if r.Method == "GET" {
38      tpl, err := template.ParseFiles("template.html")
39
40      if err != nil {
41        http.Error(w, err.Error(), http.StatusInternalServerError)
42        return
43      }
44
45      tpl.Execute(w, employees)
46      return
47    }
48
49    http.Error(w, "Invalid method", http.StatusBadRequest)
50  }
```

API #11

Sekarang mari kita buka browser dan arahkan kepada URL http://localhost:8080/employees, maka kita akan dapat melihat data employee kita seperti pada gambar di bawah.



API #12

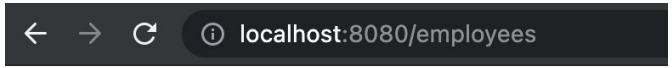
Jika kita ingin dapat melooping data employee nya, maka kita dapat melakukannya dengan menggunakan range loop seperti pada gambar di sebelah kanan.

Format pernulisan range loop nya adalah {{range $namaVariable := .}}, yang dimana cara penulisan nama variablenya masih sama seperti biasa namun harus diawali dengan tanda dolar $.

Lalu setelah itu kita juga perlu mengakhiri looping nya dengan keyword *end* yang juga perlu dimasukkan kedalam 2 tanda kurung {{end}}.

Sekarang arahkan browser kita kembali kepada URL http://localhost:8080/employees, dan kita akan melihat hasilnya seperti pada gambar kedua.

```
<html>
  <head>
    <title>HTML</title>
  </head>
  <body>
    <div>
      <ul>
        {{range $value := .}}
        <li>
          ID: {{$value.ID}}
          Name: {{$value.Name}},
          Age: {{$value.Age}},
          Division: {{$value.Division}}
        </li>
        {{end}}
      </ul>
    </div>
    </body>
</html>
```



- ID: 1 Name: Airell, Age: 23, Division: IT
- ID: 2 Name: Nanda, Age: 23, Division: Finance
- ID: 3 Name: Mailo, Age: 20, Division: IT