Scalable Web Service With Golang
sesi 8

Decoding & Parsing JSON Data

Decoding JSON To Struct #1

Pada sesi kali ini, kita akan mempelajari cara mendecode data JSON kepada sebuah struct. Caranya seperti pada gambar di sebelah kanan.

Pada line 15, kita membuah data JSON sederhana menggunakan tanda *backtick* ``. Kemudian pada line 25, kita menggunakan function *json.Unmarshal* untuk mendecode data JSON kepada struct Employee. Argumen pertama dari function *json.Unmarshal* menerima sebuah nilai dengan tipe data *slice of byte*.

Pada argument pertama itulah kita meletakkan data JSON nya tetapi harus kita ubah terlebih dahulu menjadi tipe data *slice of byte.*

Kemudian pada argumen kedua, kita meletakkan pointer dari variable *result* agar setelah data JSON berhasil di decode, datanya akan disimpan kepada variable *result*.

```
1  package main
2
3  import (
4    "encoding/json"
5    "fmt"
6  )
7
8  type Employee struct {
9    FullName string `json:"full_name"`
10   Email    string `json:"email"`
11   Age      int    `json:"age"`
12 }
13
14 func main() {
15   var jsonString = `
16     {
17       "full_name": "Airell Jordan",
18       "email": "airell@mail.com",
19       "age": 23
20     }
21   `
22
23   var result Employee
24
25   var err = json.Unmarshal([]byte(jsonString), &result)
26   if err != nil {
27     fmt.Println(err.Error())
28     return
29   }
30
31   fmt.Println("full_name:", result.FullName)
32   fmt.Println("email:", result.Email)
33   fmt.Println("age:", result.Age)
34 }
```

Decoding JSON To Struct #2

Perhatikan pada struct *Employee* yang kita telah kita buat diawal pada line 8 - 12. Terdapat sebuah format tulisan seperti `json:"full_name"`, `json:"email"`, dan `json:"age"`.

Tulisan-tulisan tersebut disebut sebagai *tag*. *Tag* kita gunakan ketika kita ingin mendecode data-data seperti JSON, form data, hingga xml kemudian kita simpan data decode tersebut kepada field-field struct nya.

Tag yang kita buat harus kita sesuaikan dengan field pada data JSONnya. Jika kita perhatikan pada line 17, terdapat field bernama *full_name*.

Sedangkan pada line 9, field pada struct *Employee* nya bernama *FullName*. Karena field *full_name* pada data JSON akan kita simpan ke pada field *FullName* pada struct *Employee,* maka kita perlu menyesuikan *tag* pada field FullName menjadi `json:"full_name"`.

```
1  package main
2
3  import (
4    "encoding/json"
5    "fmt"
6  )
7
8  type Employee struct {
9    FullName string `json:"full_name"`
10   Email    string `json:"email"`
11   Age      int    `json:"age"`
12 }
13
14 func main() {
15   var jsonString = `
16     {
17       "full_name": "Airell Jordan",
18       "email": "airell@mail.com",
19       "age": 23
20     }
21   `
22
23   var result Employee
24
25   var err = json.Unmarshal([]byte(jsonString), &result)
26   if err != nil {
27     fmt.Println(err.Error())
28     return
29   }
30
31   fmt.Println("full_name:", result.FullName)
32   fmt.Println("email:", result.Email)
33   fmt.Println("age:", result.Age)
34 }
```

Decoding JSON To Struct #3

Jika kita jalankan pada terminal kita, maka hasilnya akan seperti pada gambar di bawah.

Decoding JSON To Map

Kita juga bisa men-decode data JSON kepada sebuah tipe data *map*. Caranya seperti pada gambar di sebelah kanan.

Kita tidak perlu membuat *tag* seperti yang kita lakukan pada sebuah struct.

Jika kita jalankan, maka hasilnya akan seperti pada gambar kedua.

```go
package main

import (
  "encoding/json"
  "fmt"
)

func main() {
  var jsonString = `
    {
      "full_name": "Airell Jordan",
      "email": "airell@mail.com",
      "age": 23
    }
  `

  var result map[string]interface{}

  var err = json.Unmarshal([]byte(jsonString), &result)
  if err != nil {
    fmt.Println(err.Error())
    return
  }

  fmt.Println("full_name:", result["full_name"])
  fmt.Println("email:", result["email"])
  fmt.Println("age:", result["age"])
}
```

Decoding JSON To Map

Jika kita jalankan, maka hasilnya akan seperti pada gambar kedua.

Decoding JSON To Empty Interface

Kita juga bisa men-decode data JSON kepada sebuah tipe data *emty interface*. Caranya seperti pada gambar di sebelah kanan.

Perlu diingat disini bahwa ketika kita ingin mengakses field-fieldnya, maka harus dilakukan *type assertion* dari *empty interface* menjadi tipe data *map[string]interface{}*.

```go
package main

import (
  "encoding/json"
  "fmt"
)

func main() {
  var jsonString = `
    {
      "full_name": "Airell Jordan",
      "email": "airell@mail.com",
      "age": 23
    }
  `

  var temp interface{}

  var err = json.Unmarshal([]byte(jsonString), &temp)
  if err != nil {
    fmt.Println(err.Error())
    return
  }

  var result = temp.(map[string]interface{})

  fmt.Println("full_name:", result["full_name"])
  fmt.Println("email:", result["email"])
  fmt.Println("age:", result["age"])
}
```

Decoding JSON To Empty Interface

Jika kita jalankan, maka hasilnya akan seperti pada gambar dibawah.

Decoding JSON Array To Slice Of Struct

Kita juga bisa men-decode data *JSON array* kepada sebuah tipe data *slice of struct*. Caranya seperti pada gambar di sebelah kanan.

```go
package main

import (
  "encoding/json"
  "fmt"
)

type Employee struct {
  FullName string `json:"full_name"`
  Email    string `json:"email"`
  Age      int    `json:"age"`
}

func main() {
  var jsonString = `[
      {
        "full_name": "Airell Jordan",
        "email": "airell@mail.com",
        "age": 23
      },
      {
        "full_name": "Ananda RHP",
        "email": "ananda@mail.com",
        "age": 23
      }
    ]
  `

  var result []Employee

  var err = json.Unmarshal([]byte(jsonString), &result)
  if err != nil {
    fmt.Println(err.Error())
    return
  }

  for i, v := range result {
    fmt.Printf("Index %d: %+v\n", i+1, v)
  }
}
```

Decoding JSON Array To Slice Of Struct

Jika kita jalankan, maka hasilnya akan seperti pada gambar dibawah.

```
Index 1: {FullName:Airell Jordan Email:airell@mail.com Age:23}
Index 2: {FullName:Ananda RHP Email:ananda@mail.com Age:23}
```