HACKTIV8

Go Programming
Secure Your Go Apps

Chapter 3 Sesi 1 : GO VENDORING, HTTP REQUEST & BASIC AUTH

HACKTIV8

Go Vendoring

Go Vendoring

Vendoring di Go merupakan kapabilitas untuk mengunduh semua dependency atau 3rd party, untuk disimpan di lokal dalam folder project, dalam folder bernama vendor.

Dengan adanya folder tersebut, maka Go tidak akan lookup 3rd party ke cache folder, melainkan langsung mempergunakan yang ada dalam folder vendor. Jadi tidak perlu download lagi dari internet.

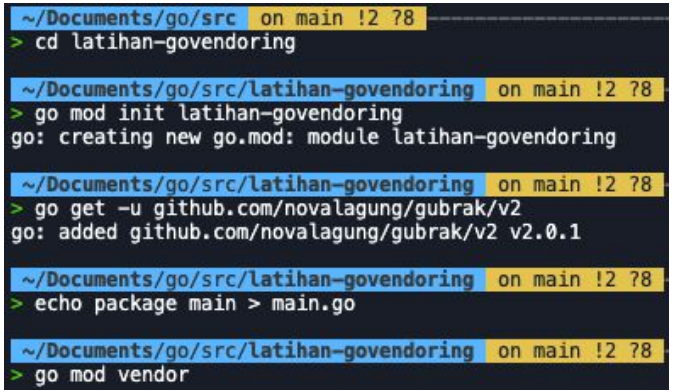
HACKTIV8

Latihan Go Vendoring

Kita akan coba praktekan untuk vendoring sebuah 3rd party bernama h8HelperRand.

Buat folder project baru dengan nama latihan-govendoring dengan isi satu file main.go. Lalu go get library h8HelperRand.

Setelah melakukan get library kita bisa buat file main.go secara manual atau mengetikan di terminal dengan echo package main > main.go. Setelah main.go sudah tergenerate kita bisa buat folder vendor dengan command go mod vendor



Latihan Go Vendoring

Lalu lihat file main.go teman-teman dan edit syntax didalamnya menjadi seperti berikut :

main.go > ...

```go
package main

import (
  "fmt"
  h8HelperRand "github.com/novalagung/gubrak/v2"
)

func main() {
  fmt.Println(h8HelperRand.RandomInt(10, 20))
}
```

HACKTIV8

Latihan Go Vendoring

Bisa dilihat, sekarang library h8HelperRand disimpan didalam folder vendor dan didalam folder github.com karena library tersebut didapatkan melalui link github.

Library tersebut juga masuk dalam folder novalagung karena didownload dari repository github novalagung mengenai library helper operation.

Berlaku juga dengan library lain yang didapatkan dari sumber lain nantinya akan tergenerate folder baru sesuai dengan sumbernya.

HACKTIV8

Latihan Go Vendoring

Untuk membuat proses build lookup ke folder vendor, kita tidak perlu melakukan apa-apa, setidaknya jika versi Go yang diinstall adalah 1.14 ke atas. Maka command build maupun run masih sama.

```
go run main.go
go build -o executable
```

Untuk yg menggunakan versi Go dibawah 1.14, disarankan untuk upgrade. Atau bisa gunakan flag -mod=vendor untuk memaksa Go lookup ke folder vendor.

```
go run -mod=vendor main.go
go build -mod=vendor -o executable
```

Manfaat vendoring adalah pada sisi kompatibilitas dan kestabilan 3rd party. Jadi dengan vendor, misal 3rd party yang kita gunakan di itu ada update yg sifatnya tidak backward compatible, maka aplikasi kita tetap aman karena menggunakan yang ada dalam folder vendor.

Jika tidak menggunakan vendoring, maka bisa saja saat go mod tidy sukses, namun sewaktu build error, karena ada fungsi yg tidak kompatibel lagi misalnya.

HTTP Request

Get Request

Sekarang kita akan mempelajari bagaimana caranya membuat sebuah Get request pada bahasa Go. Untuk membuat request tersebut, kita akan menggunakan package bernama *net/http*. Cara penerapannya seperti pada gambar disebelah kanan.

Pada line 11, kita membuat Get request dengan menggunakan function *http.Get*. Function ini menerima satu parameter dengan tipe data string yang merupakan sebuah url untuk request tujuannya. Kemudian function ini akan mereturn sebuah nilai dengan tipe data pointer dari struct *http.Response* jika request nya berhasil dan akan mereturn sebuah tipe data error yang requestnya gagal.

```go
package main

import (
  "fmt"
  "io/ioutil"
  "log"
  "net/http"
)

func main() {
  res, err := http.Get("https://jsonplaceholder.typicode.com/posts/1")
  if err != nil {
    log.Fatalln(err)
  }

  fmt.Println(res.Body)
  body, err := ioutil.ReadAll(res.Body)
  if err != nil {
    log.Fatalln(err)
  }

  defer res.Body.Close()

  sb := string(body)
  log.Println(sb)

}
```

HACKTIV8

Get Request

Pada line 17, kita mencoba untuk membaca response body dari request yang kita kirimkan dengan cara mengakses field *Body* dari struct *http.Response.* Kemudian kita menggunakan bantuan dari function *ioutil.ReadAll* agar dapat mengubah nilai yang kita akses dari field *Body* menjadi sebuah nilai dengan tipe data *slice of byte*. Lalu perlu diingat disini bahwa kita perlu menutup response body nya setelah kita selesai membacanya dengan method *Close* untuk mencegah **kebocoran memori** atau **memory leak.**

Kemudian pada line 24, kita menggunakan function *String* untuk mengubah tipe data *slice of byte* menjadi sebuah tipe data string.

```go
1  package main
2
3  import (
4    "fmt"
5    "io/ioutil"
6    "log"
7    "net/http"
8  )
9
10 func main() {
11   res, err := http.Get("https://jsonplaceholder.typicode.com/posts/1")
12   if err != nil {
13     log.Fatalln(err)
14   }
15
16   fmt.Println(res.Body)
17   body, err := ioutil.ReadAll(res.Body)
18   if err != nil {
19     log.Fatalln(err)
20   }
21
22   defer res.Body.Close()
23
24   sb := string(body)
25   log.Println(sb)
26
27 }
```

```
2021/08/15 20:24:39 {
  "userId": 1,
  "id": 1,
  "title": "sunt aut facere repellat provident occaecati excepturi optio reprehenderit",
  "body": "quia et suscipit\nsuscipit recusandae consequuntur expedita et cum\nreprehenderit molestiae ut ut quas totam\nnostrum re
rum est autem sunt rem eveniet architecto"
}
```

Post Request

Sekarang kita akan mencoba untuk membuat Post request. Contohnya seperti pada gambar disebelah kanan.

Pada line 12, kita menyiapkan data yang akan kita jadikan sebagai request body atau data yang akan dikirimkan pada Post request nya.

Pada line 18, kita merubah data yang sudah kita siapkan menjadi sebauh data JSON.

Pada line 24, kita mencoba untuk menyiapkan request nya dengan menggunakan function *http.NewRequest*.

Function ini menerima 3 parameter yaitu method dari request yang ingin kita buat, url dari request nya, dan data yang ingin kita kirim jika ada.

```go
package main

import (
  "bytes"
  "encoding/json"
  "io/ioutil"
  "log"
  "net/http"
)

func main() {
  data := map[string]interface{}{
    "title":  "Airell",
    "body":   "Jordan",
    "userId": 1,
  }

  requestJson, err := json.Marshal(data)
  client := &http.Client{}
  if err != nil {
    log.Fatalln(err)
  }

  req, err := http.NewRequest("POST", "https://jsonplaceholder.typicode.com/posts",
  bytes.NewBuffer(requestJson))
  req.Header.Set("Content-type", "application/json")
  if err != nil {
    log.Fatalln(err)
  }

  res, err := client.Do(req)
  if err != nil {
    log.Fatalln(err)
  }
  defer res.Body.Close()

  body, err := ioutil.ReadAll(res.Body)
  if err != nil {
    log.Fatalln(err)
  }

  log.Println(string(body))      Airell98, 14 months ago • initail commit …
}
```

Post Request

Perhatikan pada parameter ketiga, kita menggunakan function *bytes.NewBuffer* yang dunakan untuk mengubah tipe data dari data yang ingin kita kirim menjadi interface *io.Reader*.

Kita perlu mengubahnya menjadi interface *io.Reader* karena parameter ketiga dari function *http.NewRequest* menerima nilai dengan tipe data interface *io.Reader* dan function *bytes.NewBuffer* mereturn nilai dengan tipe data pointer dari struct *bytes.Buffer* yang telah mengimplementasikan salah satu method dari interface io.Reader.

Line 26, mengubah content type dari request header nya menjadi *application/json* karena url / api yang ingin kita tuju hanya menerima Post request dengan request header *application/json*.

Pada line 31, kita mengeksekusi requestnya dengan method *Do* yang berasal dari struct *http.Client*.

```go
package main

import (
  "bytes"
  "encoding/json"
  "io/ioutil"
  "log"
  "net/http"
)

func main() {
  data := map[string]interface{}{
    "title":  "Airell",
    "body":   "Jordan",
    "userId": 1,
  }

  requestJson, err := json.Marshal(data)
  client := &http.Client{}
  if err != nil {
    log.Fatalln(err)
  }

  req, err := http.NewRequest("POST", "https://jsonplaceholder.typicode.com/posts",
  bytes.NewBuffer(requestJson))
  req.Header.Set("Content-type", "application/json")
  if err != nil {
    log.Fatalln(err)
  }

  res, err := client.Do(req)
  if err != nil {
    log.Fatalln(err)
  }
  defer res.Body.Close()

  body, err := ioutil.ReadAll(res.Body)
  if err != nil {
    log.Fatalln(err)
  }

  log.Println(string(body))      Airell98, 14 months ago • initail commit …
}
```

Post Request

Hasil dari merunning kode disamping adalah sebagai berikut :

```
2021/08/15 21:39:35 {
  "body": "Jordan",
  "title": "Airell",
  "userId": 1,
  "id": 101
}
```

```go
package main

import (
  "bytes"
  "encoding/json"
  "io/ioutil"
  "log"
  "net/http"
)

func main() {
  data := map[string]interface{}{
    "title":  "Airell",
    "body":   "Jordan",
    "userId": 1,
  }

  requestJson, err := json.Marshal(data)
  client := &http.Client{}
  if err != nil {
    log.Fatalln(err)
  }

  req, err := http.NewRequest("POST", "https://jsonplaceholder.typicode.com/posts",
  bytes.NewBuffer(requestJson))
  req.Header.Set("Content-type", "application/json")
  if err != nil {
    log.Fatalln(err)
  }

  res, err := client.Do(req)
  if err != nil {
    log.Fatalln(err)
  }
  defer res.Body.Close()

  body, err := ioutil.ReadAll(res.Body)
  if err != nil {
    log.Fatalln(err)
  }

  log.Println(string(body))      Airell98, 14 months ago • initail commit _
}
```

HTTP Basic Auth

HACKTIV8

HTTP Basic Authentication

HTTP Basic Auth adalah salah satu teknik otentikasi 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, sebelum dimasukan ke header.

Berikut adalah contoh penulisan basic auth.

HACKTIV8

HTTP Basic Authentication

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("username:password")

// Hasilnya adalah dXNlcm5hbWU6cGFzc3dvcmQ=
```

Go menyediakan fungsi untuk meng-handle request basic auth dengan cukup mudah, jadi tidak perlu untuk memparsing header request terlebih dahulu untuk mendapatkan informasi username dan password.

HACKTIV8

Struktur Folder Project & Endpoint

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.

- main.go
- middleware.go
- student.go

Membuat Routing

Buka file 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()
}
```

Membuat Routing

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())
}
```

Membuat Routing

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)`.

HACKTIV8

Membuat Routing

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)
}
```

HACKTIV8

Student Data

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.

```go
package main

var students = []*Student{}

type Student struct {
    Id    string
    Name  string
    Grade int32
}
```

HACKTIV8

Student Data

Buat function *GetStudents*, function ini mengembalikan semua data student. Dan buat juga function **SelectStudent(id)**, function 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
}
```

Student Data

Kemudian implementasikan function *init*, buat beberapa dummy data untuk ditampung pada variabel *students*.

```
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})
}
```

Function Auth & AllowOnlyGet

Function Auth
Buka file **middleware.go**, siapkan function *Auth*.

```go
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
}
```

Function Auth & AllowOnlyGet

Tugas function *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.

Function *Auth* mengembalikan 3 informasi:

1. Username
2. Password
3. Nilai balik ke-3 ini adalah representasi valid tidak nya 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.

Function Auth & AllowOnlyGet

Function AllowOnlyGet

Function 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
}
```

HACKTIV8

Running GO With Basic Auth

Semuanya sudah siap, jalankan aplikasi.
Jangan menggunakan perintah **go run main.go**, dikarenakan dalam package **main** terdapat beberapa file lain yang harus di-ikut-sertakan pada saat runtime.

```
go run *.go
```

Test aplikasi kita kali 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
```