4 *HTTP REQUEST Pada Bahasa Pemrograman GO*

Objektif :

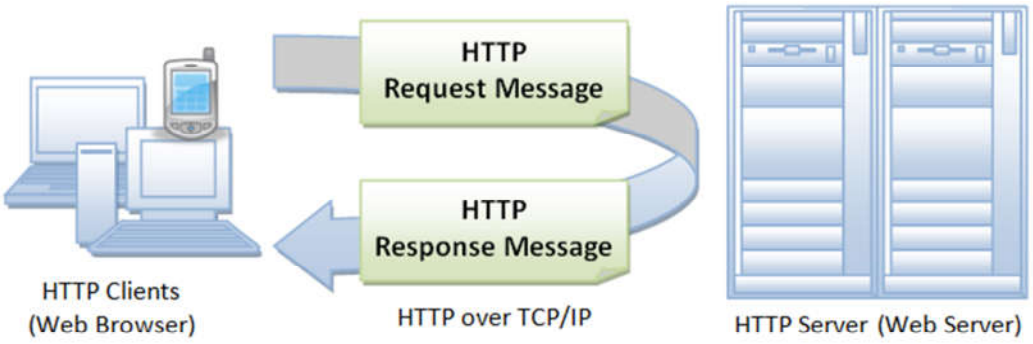
- **Mahasiswa dapat memahami konsep HTTP *Request***
- **Mahasiswa dapat mengenal beberapa metode pada HTTP *Request***
- **Mahasiswa dapat memahami bagaimana implementasi HTTP *Request* pada bahasa pemrograman Golang**

4.1 Sekilas Tentang *HTTP*

Hypertext Transfer Protocol atau *HTTP* merupakan bagian kehidupan dari sebuah *website* yang berupa protokol aplikasi untuk sistem informasi terdistribusi, kolaboratif, dan *hypermedia* sebagai dasar komunikasi data untuk *World Wide Web (WWW)* yang berbentuk teks terstruktur dan menggunakan *hyperlink* antar node yang mengandung sebuah teks. *HTTP* memungkinkan untuk berkomunikasi antara klien dan *server* pada suatu *website* dalam bentuk *request* (permintaan) dan *response* (tanggapan).

HTTP memiliki fungsi untuk menentukan bagaimana pesan, dokumen atau data dapat ditransmisikan atau diformat menjadi bentuk lain yang dapat diterima *browser*. Sehingga semua data yang diinginkan oleh klien bisa di akses atau ditampilkan.

Pada dasarnya cara kerja *HTTP* cukup sederhana, saat sebuah website jalankan melalui *browser* maka *HTTP* akan bertugas untuk menghubungkannya dengan *World Wide Web (WWW)*. Selanjutnya apabila terdapat sebuah *request* maka *HTTP* yang akan menjadi perantara pada *request* tersebut kepada *server* yang kemudian akan ditanggapi dalam bentuk *response.*



Gambar 4.1 Ilustrasi komunikasi antara *HTTP* klien dengan server

4.2 *HTTP Request*

HTTP memiliki beberapa metode *request* yang berbeda sesuai dengan tipe *request* yang dilakukan. Berdasarkan spesifikasi *HTTP* 1.1 terdapat 8 buah metode *request* pada *HTTP* yaitu *GET, POST, PUT, DELETE, HEAD, OPTIONS, TRACE, CONNECT*. Diantara 8 metode *request* tersebut, terdapat 4 metode yang umum digunakan yaitu *GET, POST, PUT* dan *DELETE*. Berikut merupakan penjelasan dari 4 metode tersebut:

- *GET*

  GET merupakan *method* yang paling umum dan paling banyak digunakan. *GET* digunakan untuk meminta data tertentu dari server. *HTTP*/1.1 mewajibkan seluruh *web server* untuk mengimplementasikan *method* ini.

- *POST*

  POST digunakan untuk mengirimkan data masukan pengguna ke *server*. Pada prakteknya, *POST* digunakan untuk mengambil data yang dimasukkan dari *form HTML*. Data yang diisikan pada form biasanya dikirimkan kepada *server*, dan *server* kemudian menentukan ke mana data akan dikirimkan selanjutnya (misal: disimpan ke *database* atau diproses oleh aplikasi lain).

- *PUT*

  Method PUT menuliskan data pada *server*, sebagai kebalikan dari *GET* yang membaca data dari *server*. Beberapa sistem publikasi atau manajemen konten memungkinkan untuk membuat sebuah halaman *web* baru pada *web server* dengan menggunakan *PUT*.

  Cara kerja *PUT* sangat sederhana, yaitu server membaca isi *Entity Body* dari *request* dan menggunakan isi tersebut untuk membuat halaman baru pada *URL* yang diberikan *request*, atau jika *URL* telah ada maka isi data akan dierbaharui. Karena *PUT*

menggantikan data yang ada pada *server*, kebanyakan *server* biasanya mewajibkan autentikasi sebelum melakukan *PUT*.

- *DELETE*

  DELETE, seperti namanya, meminta server untuk menghapus data yang ada pada server, sesuai dengan *URL* yang diberikan oleh *client*. Begitupun, *client* tidak diberi jaminan bahwa penghapusan akan dijalankan ketika menggunakan *DELETE*. Hal ini terjadi karena standar *HTTP* memperbolehkan server untuk menolak atau mengabaikan permintaan *DELETE* dari *client*.

Pada metode *HTTP* terdapat beberapa istilah yang menggambarkan sifat dari masing-masing metode. Berikut merupakan istilah dari penjelasan singkat dari masing-masing istilah tersebut :

- ***Safe Method***

Merupakan istilah untuk metode yang aman digunakan terus menerus. Yang dimaksud aman adalah berapa banyaknya pun dilakukan *request* terhadap metode tersebut maka tidak akan mengubah data pada *server*. Beberapa metode yang termasuk *safe method* diantaranya yaitu: *HEAD, GET, OPTIONS* dan *TRACE.*

- ***Idempotent Method***

Merupakan istilah untuk metode yang dilakukan berulang-ulang hanya akan menyebabkan perubahan pada *server* sebanyak satu kali (pada request pertama kali). *Idempotent Method* sangat kontras dengan missal metode *POST*. Metode *POST* akan terus melakukan penambahan pada *database* apabila terus dilakukan *request* yang sama. Oleh sebab itu apabila sedang mengirim menggunakan metode *POST* dan dapat me-*refresh* halaman, maka akan muncul kotak dialog konfirmasi pada *browser* bahwa akan mengirim ulang *request POST* yang sama sebelumnya pada *browser*. Beberapa metode yang termasuk kedalam *Idempotent Method* diantaranya yaitu: *POST, PUT* dan *DELETE.*

4.3 Kode Status *HTTP*

Pada saat melakukan *HTTP request* maupun *response*, berhasil atau tidaknya *request* atau *response* yang dilakukan itu ditandai dengan sebuah status dari *HTTP* itu sendiri yang berupa kode. Kode status pada *HTTP* terbagi ke dalam 5 kategori besar yang diantaranya seperti tabel dibawah ini.

Tabel 4.1 Kategori Kode Status *HTTP*

Keseluruhan Kode	Kode yang Terdefinisi	Kategori
100-199	100-101	Informasional
200-299	200-206	Sukses
300-399	300-305	*Redirection*
400-499	400-415	Kesalahan *Client*
500-599	500-505	Kesalahan *Server*

Adapun tabel di bawah ini menunjukkan seluruh kode pada status HTTP yang ada, sesuai dengan spesifikasi HTTP/1.1.

Tabel 4.2 Kode Status *HTTP*

Kode Status	***Reason Phrase***	Keterangan
100	*Continue*	Bagian awal dari *request* telah diterima. Lanjutkan pemrosesan.
101	*Switching Protocol*	*Server* beralih menggunakan protocol yang berbeda, sesuai permintaan *client*.
200	*OK*	*Request* berjalan sukses
201	*Created*	Data berhasil dibuat (untuk *request* yang membuat objek baru).
202	*Accepted*	*Request* diterima, tetapi *server* tidak melakukan apa-apa.
203	*Non-Authoritative Information*	Transaksi berhasil, tetapi informasi pada *header* dari *response* tidak berasal dari *server* asli, tapi dari kopi data.
204	*No Content*	*Response* hanya berisi *header* dan status *line*, tanpa *body*.
205	*Reset Content*	Kode yang dirancang khusus untuk *browser*; meminta *browser* menghapus isi dari semua elemen *form HTML* pada halaman aktif.
206	*Partial Content*	*Request* parsial berhasil.
300	*Multiple Choices*	*Client* memberikan *request URL* yang merujuk ke beberapa data. Kode dikirimkan dengan beberapa pilihan yang dapat dipilih *client.*

301	*Moved Permanently*	*URL* dari *request* telah dipindahkan. *Response* wajib berisi lokasi *URL* baru.
302	*Found*	Sama seperti 301, tetapi perpindahan hanya untuk sementara. *URL* data sementara diberikan melalui *Header Location*.
303	*See Other*	*Response* dari *request* ada pada halaman lain, yang harus diambil dengan *GET*. Jika *request* merupakan *POST, PUT*, atau *DELETE,* asumsikan server telah menerima data dan *redirect* dilakukan dengan *GET* yang baru.
304	*Not Modified*	Mengindikasikan tidak ada perubahan data dari *request* sebelumnya.
305	*Use Proxy*	Data harus diakses dari *proxy*. Lokasi *proxy* diberikan di *Header Location.*
306	*Switch Proxy*	Kode status tidak lagi digunakan untuk sekarang.
307	*Temporary Redirect*	*Request* harus dilakukan kembali pada *URI* lain untuk sementara. *Server* dianggap tidak menerima data, sehingga jika *request* awal merupakan *POST* maka *request* baru juga harus *POST.*
400	*Bad Request*	*Client* memberikan *request* yang tidak lengkap atau salah.
401	*Unauthorized*	Akses ditolak. *Header* autentikasi akan dikirimkan.
402	*Payment Required*	Belum digunakan, tetapi disiapkan untuk penggunaan pada masa depan.
403	*Forbidden*	*Request* ditolak oleh *server*.
404	*Not Found*	*URL* yang diminta tidak ditemukan pada *server*.
405	*Method Not Allowed*	*Method* tidak didukung *server*. *Header* daftar *method* yang didukung untuk *URL* tersebut harus ada pada *response*.
406	*Not Acceptable*	Jika *client* memberikan daftar format yang

		dapat dibaca, kode ini menandakan *server* tidak dapat memberikan data dalam format itu.
407	*Proxy Authentication Required*	Seperti 401, tetapi autentikasi dilakukan terhadap *proxy*.
408	*Request Timeout*	*Client* terlalu lama dalam menyelesaikan *request.*
409	*Conflict*	*Request* menyebabkan konflik pada data yang diminta.
410	*Gone*	Seperti 404, tetapi *server* pernah memiliki data tersebut.
411	*Length Required*	*Request* harus memiliki *header Content-Length.*
412	*Precondition Failed*	Diberikan jika client membuat *conditional request* dan kondisi tidak terpenuhi.
413	*Request Entity Too Large*	*Entity* pada *request* terlalu besar.
414	*Request URI Too Long*	*URI* yang dikirimkan kepada *server* terlalu panjang.
415	*Unsupported Media Type*	Format data yang dikirimkan tidak didukung oleh *server.*
416	*Request Range Not Satisfiable*	Tidak dapat memenuhi cakupan data, untuk *request* yang menintanya.
417	*Expectation Failed*	Server tidak dapat memenuhi parameter *Expectation* yang ada pada *request*.
500	*Internal Server Error*	*Server* mengalami error ketika memproses *request*.
501	*Not Implemented*	*Client* membuat *request* yang di luar kemampuan *server*.
502	*Bad Gateway*	*Server proxy* atau *gateway* menemukan *response* yang aneh dari titik berikutnya pada rantai *response*.
503	*Service Unavailable*	*Server* untuk sementara tidak dapat memproses *request*.
504	*Gateway Timeout*	Sama dengan 408 tetapi dari *gateway* atau *proxy*, bukan *client.*

505	*HTTP Version Not Supported*	Versi protokol tidak didukung oleh *server*.

4.4 Implementasi HTTP Request

Sebelum melakukan implementasi *HTTP request* pada bahasa pemrograman GO, pastikan sudah membuat file bernama **main.go.** Setelah itu pada **main.go** yang telah dibuat ketikan kode seperti berikut ini.

```
1   package main
2
3   import (
4       "net/http"
5       "log"
6       "encoding/json"
7   )
8
9   type siswa struct {
10      ID    string
11      Nama  string
12      Umur int
13  }
14
15  var data_siswa = []siswa{
16      siswa{"A001", "Jojon", 80},
17      siswa{"A002", "Riko", 85},
18      siswa{"A003", "Bambang", 90},
19      siswa{"A004", "Subagja", 70},
20  }
21
```

Gambar 4.2 Kode main.go

- Library yang digunakan

1. ***net/http***

   Package ini, selain berisikan *tools* untuk keperluan pembuatan *web*, juga berisikan fungsi-fungsi untuk melakukan *HTTP request*.

2. **log**

   Package ini menyediakan fungsi untuk menampilkan sebuah *log*.

3. ***encoding/json***

   Package ini berisikan banyak fungsi untuk kebutuhan operasi *json*.

- *Struct siswa* diatas digunakan sebagai tipe elemen *array* sebuah sampel data, ditampung variabel *data_siswa*.

- Fungsi ***Post***

  Pada gambar 4.3 dapat melihat potongan kode berupa fungsi untuk melakukan *request* dengan menggunakan *method POST*. Pada fungsi tersebut terdapat 2 parameter yaitu *response* dengan inisial **w** dan *request* dengan inisial **r.** Didalam fungsi ini dilakukan yang namanya pengecekan kondisi *method* yang digunakan yaitu *POST*, apabila kondisi *method* tersebut terpenuhi maka akan menampilkan data_siswa dalam bentuk *json*. Namun apabila *method* yang digunakan tidak sesuai maka akan menampikan sebuah pesan *error* "Jenis method tidak ditemukan".

```
22  func Post(w http.ResponseWriter, r *http.Request) {
23      w.Header().Set("Content-Type", "application/json")
24
25      if r.Method == "POST" {
26          var result, err = json.Marshal(data_siswa)
27
28          if err != nil {
29              http.Error(w, err.Error(), http.StatusInternalServerError)
30              return
31          }
32          w.Write(result)
33      }
34      http.Error(w, "Jenis method tidak ditemukan", http.StatusBadRequest)
35      return
36  }
37
```

Gambar 4.3 Fungsi *POST*

- Fungsi ***Get***

  Pada Gambar 4.4 terdapat potongan kode berupa fungsi untuk melakukan *request* dengan menggunakan *method GET*. Pada fungsi tersebut terdapat 2 parameter yaitu *response* dengan inisial **w** dan *request* dengan inisial **r.** Didalam fungsi ini dilakukan yang namanya pengecekan kondisi *method* yang digunakan yaitu *GET*, apabila kondisi *method* tersebut terpenuhi maka akan menampilkan data_siswa dalam bentuk *json*. Berbeda dengan fungsi *Post* sebelumnya, untuk menampilkan *json* data_siswa pada fungsi *Get* ini memerlukan sebuah id pada *url*-nya. Namun apabila *method* yang

digunakan tidak sesuai maka akan menampikan sebuah pesan *error* "Jenis method tidak ditemukan".

```
38  func Get(w http.ResponseWriter, r *http.Request) {
39      w.Header().Set("Content-Type", "application/json")
40
41      if r.Method == "GET" {
42          queryValues := r.URL.Query()
43          id  := queryValues.Get("id")
44          var result []byte
45          var err error
46
47          for _, each := range data_siswa {
48              if each.ID == id {
49                  result, err = json.Marshal(each)
50
51                  if err != nil {
52                      http.Error(w, err.Error(), http.StatusInternalServerError)
53                  }
54                  w.Write(result)
55              }
56          }
57          http.Error(w, "Data tidak ditemukan", http.StatusBadRequest)
58      }
59      http.Error(w, "", http.StatusBadRequest)
60  }
61
```

Gambar 4.4 Fungsi *GET*

- **Fungsi *Delete***

  Pada Gambar 4.5 terdapat potongan kode berupa fungsi untuk melakukan *request* dengan menggunakan *method DELETE*. Pada fungsi tersebut terdapat 2 parameter yaitu *response* dengan inisial **w** dan *request* dengan inisial **r.** Didalam fungsi ini dilakukan yang namanya pengecekan kondisi *method* yang digunakan yaitu *DELETE*, apabila kondisi *method* tersebut terpenuhi maka akan menampilkan *interface* yang berisi data_siswa, pesan dan *status request* dalam bentuk *json*. Namun apabila *method* yang digunakan tidak sesuai maka akan menampikan sebuah pesan *error* "Jenis method tidak ditemukan".

```
62  func Delete(w http.ResponseWriter, r *http.Request) {
63      w.Header().Set("Content-Type", "application/json")
64      if r.Method == "DELETE" {
65          responseWithMap := map[string]interface{}{
66              "data" : data_siswa,
67              "pesan" : "Method Delete sedang berjalan",
68              "status" : http.StatusOK,
69          }
70          json.NewEncoder(w).Encode(responseWithMap)
71      }
72      http.Error(w, "Jenis method tidak ditemukan", http.StatusBadRequest)
73      return
74  }
75
```

Gambar 4.5 Fungsi *DELETE*

- Fungsi ***Put***

  Pada Gambar 4.6 terdapat potongan kode berupa fungsi untuk melakukan *request* dengan menggunakan *method PUT*. Pada fungsi tersebut terdapat 2 parameter yaitu *response* dengan inisial **w** dan *request* dengan inisial **r.** Didalam fungsi ini dilakukan yang namanya pengecekan kondisi *method* yang digunakan yaitu PUT, apabila kondisi *method* tersebut terpenuhi maka akan menampilkan *interface* yang berisi data_siswa, pesan dan *status request* dalam bentuk *json*. Namun apabila *method* yang digunakan tidak sesuai maka akan menampikan sebuah pesan *error* "Jenis method tidak ditemukan".

```
76  func Put(w http.ResponseWriter, r *http.Request) {
77      w.Header().Set("Content-Type", "application/json")
78      if r.Method == "PUT" {
79          responseWithMap := map[string]interface{}{
80              "data" : data_siswa,
81              "pesan" : "Method PUT sedang berjalan",
82              "status" : http.StatusOK,
83          }
84          json.NewEncoder(w).Encode(responseWithMap)
85      }
86      http.Error(w, "Jenis method tidak ditemukan", http.StatusBadRequest)
87      return
88  }
89
```

Gambar 4.6 Fungsi *PUT*

- Fungsi ***main***

  Pada gambar 4.7, dalam sebuah proyek harus ada *file* program yang berisikan sebuah fungsi bernama *main* . Fungsi tersebut harus berada dalam package yang juga bernama main . Fungsi main adalah yang dipanggil pertama kali pada saat program dijalankan.

  Didalam fungsi *main* tersebut terdapat fungsi *http.HandleFunc()* yang digunakan untuk melakukan ***routing*** pada aplikasi web. Maksud dari *routing* adalah penentuan aksi ketika *url* tertentu diakses oleh *user*. Kemudian terdapat fungsi *http.ListenAndServe()* yang digunakan untuk menghidupkan *server* sekaligus menjalankan aplikasi menggunakan *server* tersebut. Yang artinya program tersebut dapat di akses di port :8088.

```
90   func main() {
91       http.HandleFunc("/get", Get)
92       http.HandleFunc("/post", Post)
93       http.HandleFunc("/del", Delete)
94       log.Println("localhost : 8088")
95       http.ListenAndServe(":8088", nil)
96   }
```

Gambar 4.7 Fungsi *main*

- Menjalankan file main.go

```
PS C:\laragon\www\go\src\go-learn\Praktik\BAB_B> go run main.go
2019/09/21 22:04:52 localhost : 8088
```

Gambar 4.8 Menjalankan *File main.go*

Untuk menjalankan program yang telah dibuat sebelumnya yaitu dengan cara seperti pada Gambar 4.8, dengan menggunakan perintah ***go run nama_file.go.*** Pada saat menjalankannya pastikan *path* dari lokasi file **main.go** sesuai dengan yang ada pada *cmd*. Maka akan muncul tulisan *localhost : 8088* yang telah ditentukan pada fungsi *main* sebelumnya yaitu untuk mengakses program tersebut pada *port :8088* dengan alamat *http://localhost:8088* atau *http://127.0.0.1:8088*.