MATERI

Hari 14 - Web Service API Server & Middleware

Web Service API Server

Web Service API adalah sebuah web yang menerima request dari client dan menghasilkan response, biasa berupa JSON/XML.

Membuat API dengan Method `GET`

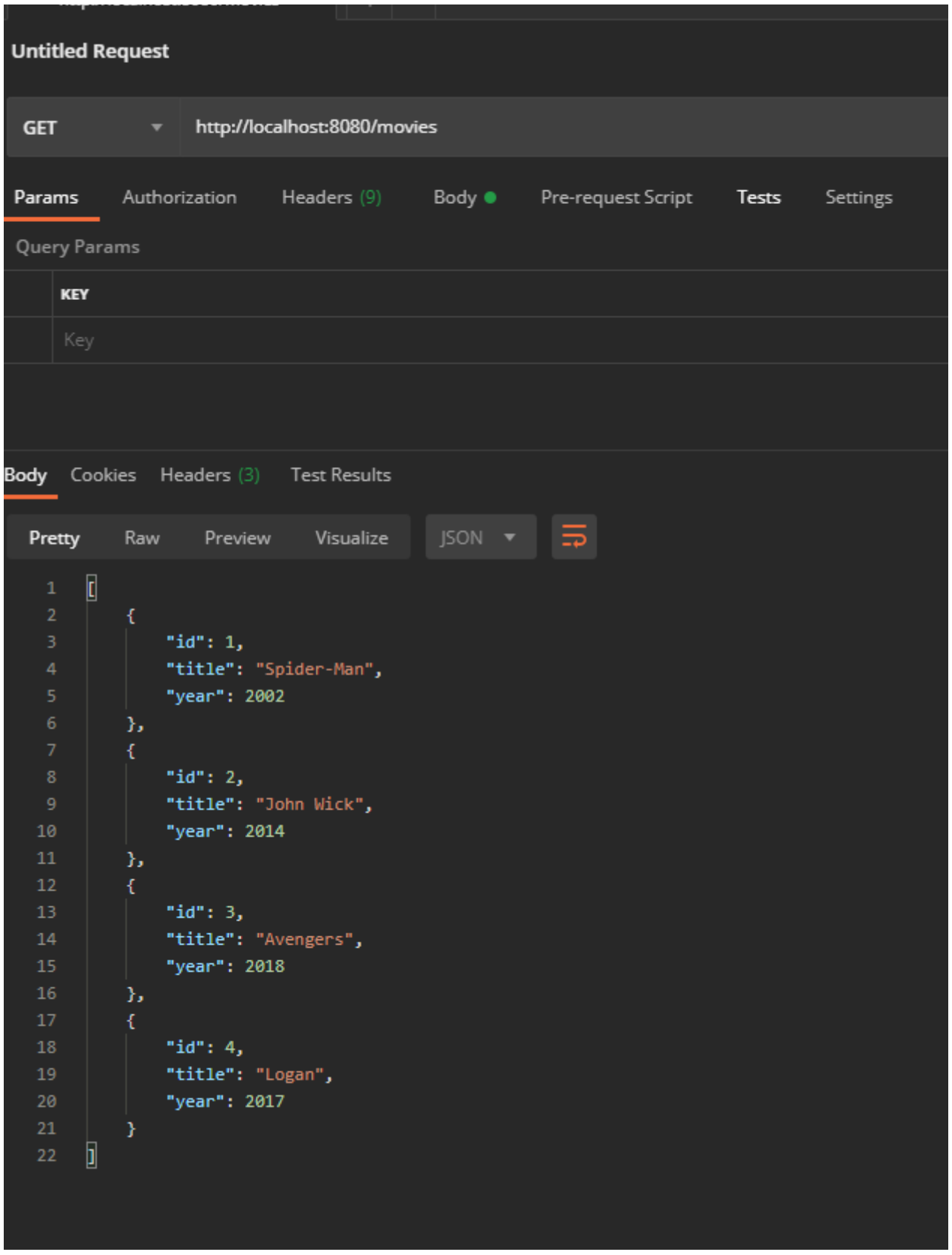
Method `GET` paling sering digunakan untuk menampilkan sebuah data **json**. Maka saya akan berikan contoh pertama dengan method **GET**.

Method GET dapat di akses melalui browser atau tools api lainnya. Selain itu juga dapat di ambil dari website lainnya, dengan catatan harus mengaktifkan **cors**.

Yuk langsung saja kita buat.

```
package main

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

type Movie struct {
  ID    int    `json:"id"`
  Title string `json:"title"`
  Year  int    `json:"year"`
}

func Movies() []Movie {
  movs := []Movie{
    {1, "Spider-Man", 2002},
    {2, "John Wick", 2014},
    {3, "Avengers", 2018},
    {4, "Logan", 2017},
  }
  return movs
}

// GetMovies
func getMovies(w http.ResponseWriter, r *http.Request) {
  if r.Method == "GET" {
    movies := Movies()
    dataMovies, err := json.Marshal(movies)

    if err != nil {
      http.Error(w, err.Error(), http.StatusInternalServerError)
    }

    w.Header().Set("Content-Type", "application/json")
    w.WriteHeader(http.StatusOK)
    w.Write(dataMovies)
    return
  }
  http.Error(w, "ERROR....", http.StatusNotFound)
}

func main() {
  http.HandleFunc("/movies", getMovies)
  fmt.Println("server running at http://localhost:8080")

  if err := http.ListenAndServe(":8080", nil); err != nil {
    log.Fatal(err)
  }
}
```

Setelah web server sudah berjalan, web service yang telah dibuat perlu untuk di-tes. Salah satu tools yang digunakan untuk mengecek web service API adalah postman



Perhatikan kode di atas, terdapat `http server` yang mengalankan fungsi `GetMovies()`. Dimana di dalam fungsi tersebut akan memeriksa dahulu method yang di akses, apabila method itu GET maka akan di teruskan ke baris kode berikutnya.

Kode baris berikutnya memanggil method `Movies()`, dimana nilai balik dari fungsi tersebut adalah data array dari struct `Movie`. Setelah itu maka data tersebut di ubah format ke dalam tipe byte dengan sintaks `marshal`.

Untuk menampilkan ke halaman browser menggunakan kode `w.Write(dataMovie)`. Agar Format JSON harus menambahkan sintaks `w.Header().Set("Content-Type", "application/json")`.

Sedangkan untuk sintaks `w.WriteHeader(http.StatusOK)` digunakan untuk memeberikan kode status header. Jika belum paham mengenai kode status http bisa membaca di https://restfulapi.net/http-status-codes/.

Membuat API dengan Method `POST`

Kalau menggunakan Method `POST` ini tidak bisa langsung di akses di browser, tapi dapat menggunakan form `HTML` atau `Postman` sebagai tools. Di karenakan kita tidak membuat website namun membuat API Web Service dengan Golang maka untuk Uji Coba saya menggunakan `Postman`.

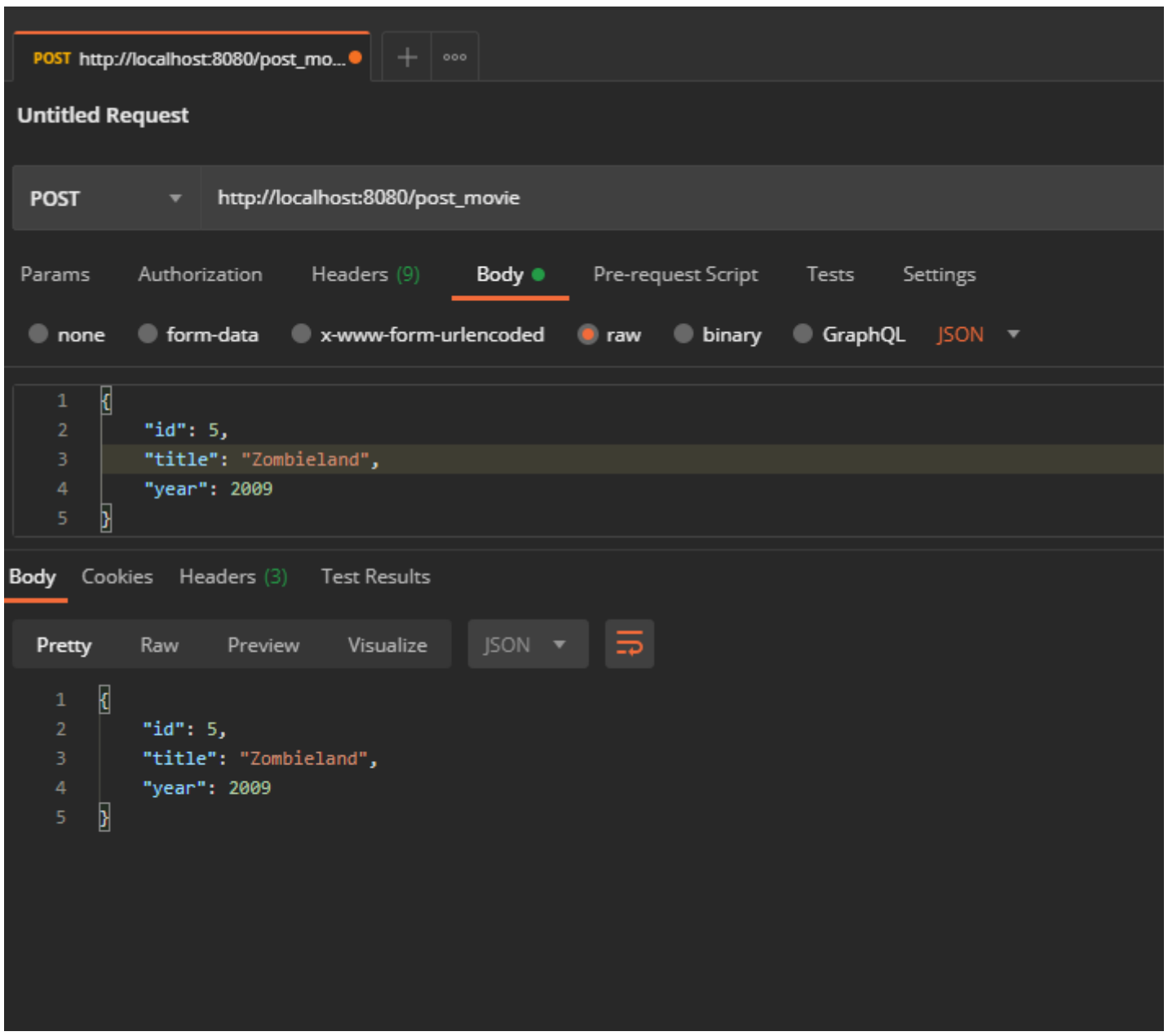
Sebelum ke kode program ada hal yang perlu kita pahami tentang input data post. Secara umum dan sering di pakai dapat menggunakan 2 macam cara untuk menerima data yaitu dengan `form` dan `teks json`, kedua hal ini biasa di sebut `content type`. Cara menerima datanya pun berbeda.

Langsung saja kita buat contoh kode golang nya.

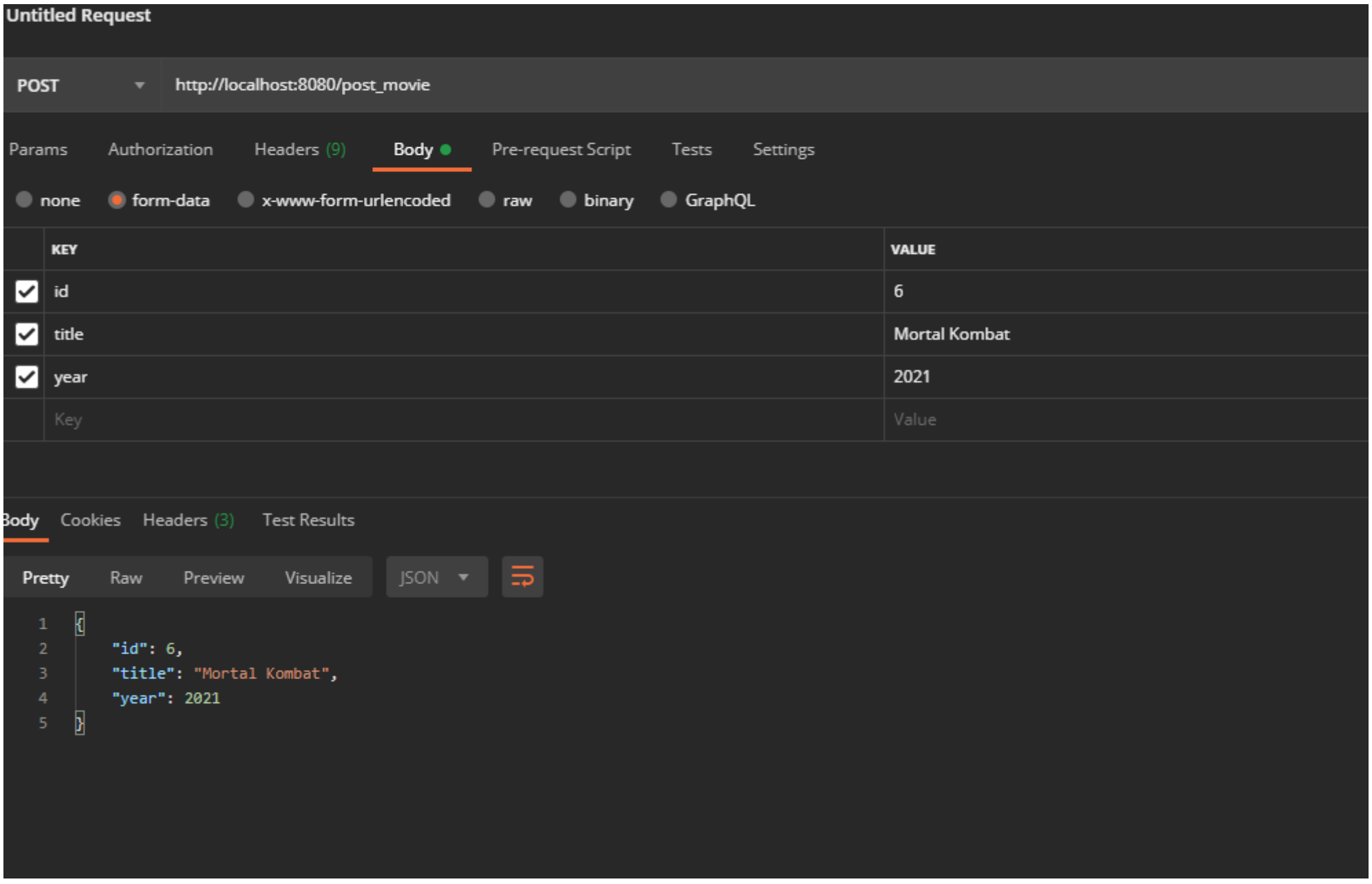
```
var Mov Movie
if r.Method == "POST" {
  if r.Header.Get("Content-Type") == "application/json" {
    // parse dari json
    decodeJSON := json.NewDecoder(r.Body)
    if err := decodeJSON.Decode(&Mov); err != nil {
      log.Fatal(err)
    }
  } else {
    // parse dari form
    getID := r.PostFormValue("id")
    id, _ := strconv.Atoi(getID)
    title := r.PostFormValue("title")
    getYear := r.PostFormValue("year")
    year, _ := strconv.Atoi(getYear)
    Mov = Movie{
       ID:    id,
       Title: title,
       Year:  year,
    }
  }

  dataMovie, _ := json.Marshal(Mov) // to byte
  w.Write(dataMovie)                // cetak di browser
  return
 }

 http.Error(w, "NOT FOUND", http.StatusNotFound)
 return
}

// pada function main

http.HandleFunc("/post_movie", PostMovie)
```

Percobaan pada kode diatas dilakukan 2 kali yaitu dengan menggunakan `raw data json` dan `form-data`.

Uji coba dengan JSON



Uji coba dengan form-data



Routing `"/post_movie"` mengakses fungsi dengan nama `PostMovie`. Sama seperti hal nya dengan method GET untuk memeriksa method yang di akses, namun di ganti dengan method `POST`.

Untuk menangkap tipe konten yang di inputkan dapat menggunakan kode `r.Header.Get("Content-Type") `. Jika tipe tersebut json untuk menerima data harus menggunakan tipe data maupun struct di tandai dengan kode `decodeJSON := json.NewDecoder(r.Body)` dan `decodeJSON.Decode(&Mov)` agar masuk struct dengan nama Mahasiswa.

Jika data tersebut bukan json maka untuk mengambil data nya dapat menggunakan sintaks `PostFormValue` dan di dalam nya di tulis nama `field` nya. Data kembalian terhadap fungsi itu yaitu `string`, apabila ingin mengubah ke bentuk tipe data `integer` dapat menggunakan manipulasi string yaitu `Atoi`.

Middleware

Middleware adalah satu satu mekanisme keamanan website. Middleware berjalan sebelum fungsi utama di proses.

Sebagai contoh serderhana, misalnya terdapat routing untuk mengarahkan ke dalam fungsi, fungsi tersebut digunakan untuk melihat data. Nah agar fungsi tersebut tidak dapat di akses orang maka harus melewati `middleware`, middleware berperan untuk menghentikan atau meneruskan proses.

Contoh paling simpel isi dari middleware adalah pemeriksaan suatu teks yang bisa di kirim melalui body dan header melalui `http request`.

Untuk the real word projek middleware di gunakan untuk auntentifikasi login menggunakan token, middleware berperan untuk memeriksa apakah token yang dikirim sesuai atau tidak.

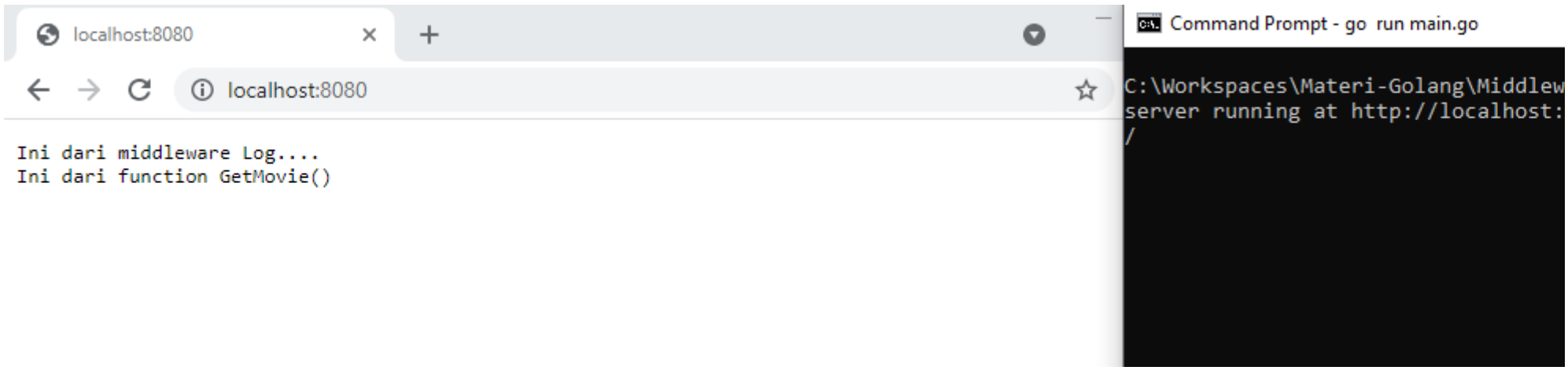
Mari kita coba buat middleware sederhana.

Membuat Middleware

Sekarang kita coba latihan menggunakan middleware, study case kali ini akan memanfaatkan middleware untuk melakukan `logging`, artinya untuk mencatat **URL** yang di akses.

```
  "fmt"
  "net/http"
)

// Fungi Log yang berguna sebagai middleware
func log(next http.Handler) http.Handler {
  return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintf(w, "Ini dari middleware Log....\n")
    fmt.Println(r.URL.Path)
    next.ServeHTTP(w, r)
  })
}

// Fungsi GetMovie untuk mengampilkan text string di browser
func GetMovie(w http.ResponseWriter, r *http.Request) {
  w.Write([]byte("Ini dari function GetMovie()"))
}

func main() {
  // konfigurasi server
  server := &http.Server{
    Addr: ":8080",
  }

  // routing
  http.Handle("/", log(http.HandlerFunc(GetMovie)))

  // jalankan server
  fmt.Println("server running at http://localhost:8080")
  server.ListenAndServe()
}
```

Hasilnya :



Perhatikan kode di atas, terdapat http server yang berjalan di port `8000`. Hanya memiliki satu routing yaitu “/”, dimana routing tersebut digunakan untuk menjalankan fungsi `GetMovie()`. Namun jika di lihat dari kode di atas, bahwa GetMovie di bungkus dengan fungsi `log`.

Sekarang perhatikan fungsi `log`. Di log ada parameter `next http.Handler`. Parameter itu merupakan sebuah interface di Go. Interface Handler memiliki method yaitu `ServeHTTP`. Di dalam Method `ServerHTTP` terdapat parameter `ResponseWriter, *Request`. Yang artinya untuk menggunakan `http.handler` harus di bungkus `http.HandlerFunc`.

Menggunakan Middleware untuk Memeriksa Parameter URL

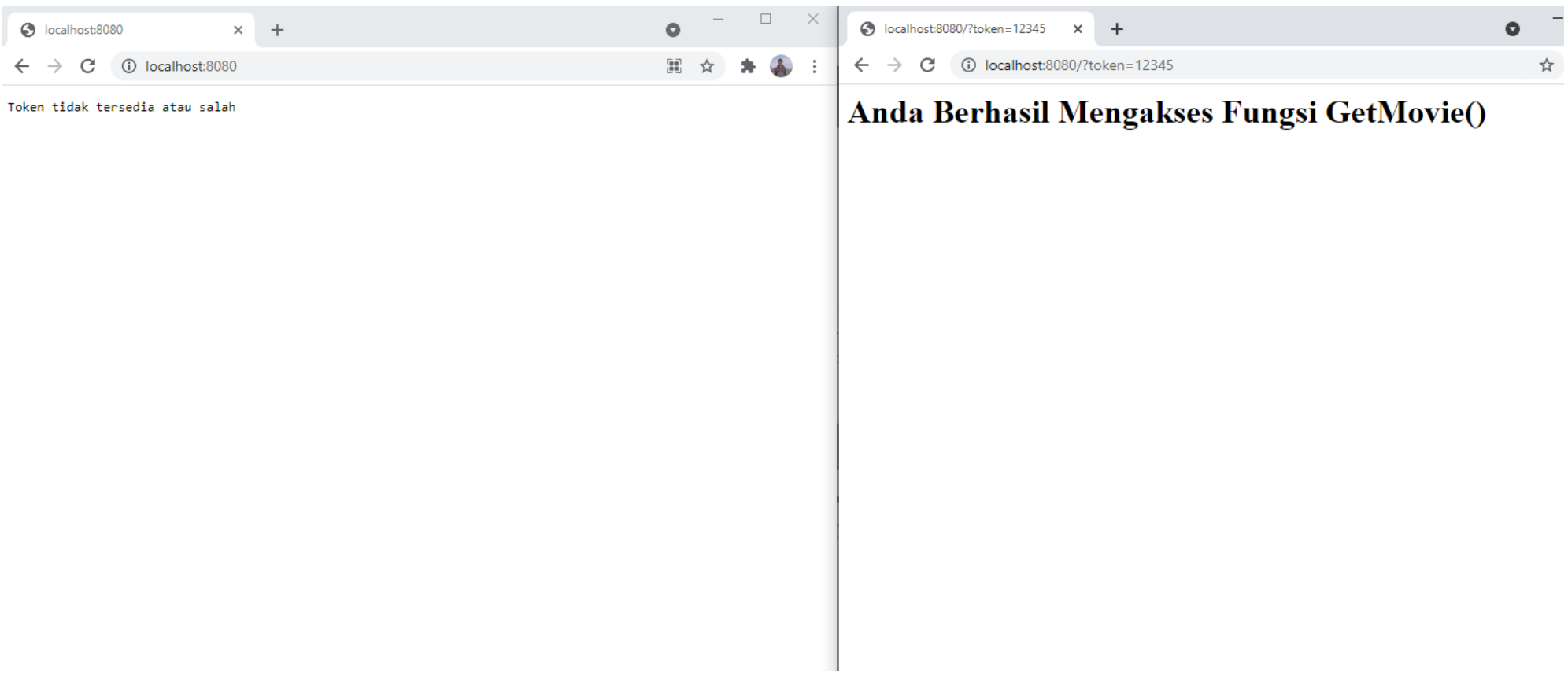
Sekarang kita buat contoh bagaimana cara melakukan pemeriksaan nilai yang di kirim dari URL.

Studi kasus kali ini yaitu mengambil parameter dan di lakukan pengecekan, apabila sukses akan di lanjutkan tapi kalau tidak akan di hentikan.

```
    "fmt"
    "net/http"
)

// Fungi CekLogin yang berguna sebagai middleware
func CekLogin(next http.Handler) http.Handler {
    return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {

        if r.URL.Query().Get("token") != "12345" {
            fmt.Fprintf(w, "Token tidak tersedia atau salah\n")
            return
        }
        next.ServeHTTP(w, r)
    })
}

// Fungsi GetMovie untuk mengampilkan text string di browser
func GetMovie(w http.ResponseWriter, r *http.Request) {
    w.Write([]byte("<h1>Anda Berhasil Mengakses Fungsi GetMovie() </h1>"))
}

func main() {

    // konfigurasi server
    server := &http.Server{
        Addr: ":8080",
    }
    // routing
    http.Handle("/", CekLogin(http.HandlerFunc(GetMovie)))

    // jalankan server
    fmt.Println("server running at http://localhost:8080")
    server.ListenAndServe()
}
```

Perhatikan kode yang ada di dalam fungsi `CekLogin()`. Di fungsi tersebut terdapat kode `r.URL.Query().Get("token") != "12345"`. Kode tersebut untuk mengambil parameter `token` dari URL. Ketika token tersebut bernilai 12345 maka akan di teruskan ke fungsi `GetMovie()` Tapi jika tidak akan muncul keterangan saja dan tidak di ijinkan mengakses fungsi `GetMovie()`.

Hasilnya :



Bisa di lihat pada 2 gambar di atas, dimana gambar sebelah kiri parameter yang di berikan di URL benar, sedangkan di sebelah kanan tidak menggunakan parameter.

Menerapkan Middleware untuk Basic Authentication API

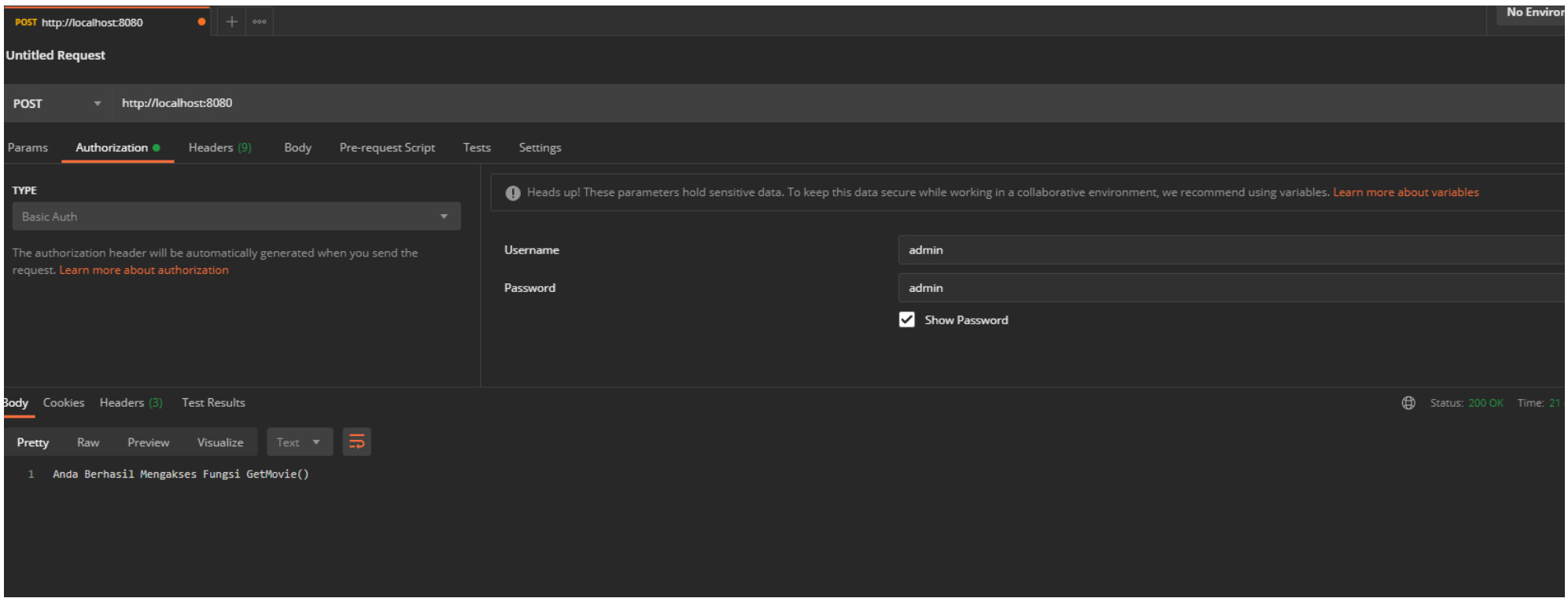
Di dalam projek nyata penggunaan Authentication dapat di gunakan untuk melakukan login dimana dapat menggunakan Authentication untuk memeriksa username dan password.

Basic auth merupakan salah satu jenis Authentication yang paling mudah digunakan dimana untuk melakukannya kita harus memberikan nilai **username** dan **password**. Hal ini biasa nya untuk **Authentication** kebutuhan **API**.

Berhubung tidak ada front end yang dapat digunakan untuk menguji, pada tutorial golang kali ini uji coba dapat di lakukan menggunakan tools Postman

```
import (
  "net/http"
)

// Fungi Log yang berguna sebagai middleware
func Auth(next http.Handler) http.Handler {
  return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
    uname, pwd, ok := r.BasicAuth()
    if !ok {
      w.Write([]byte("Username atau Password tidak boleh kosong"))
      return
    }

    if uname == "admin" && pwd == "admin" {
      next.ServeHTTP(w, r)
      return
   }
    w.Write([]byte("Username atau Password tidak sesuai"))
    return
  })
}

// Fungsi GetMovie untuk mengampilkan text string di browser
func GetMovie(w http.ResponseWriter, r *http.Request) {
  if r.Method == "POST" {
    w.Write([]byte("<h1>Anda Berhasil Mengakses Fungsi GetMovie() </h1>"))
  }
}

func main() {
  // konfigurasi server
  server := &http.Server{
    Addr: ":8080",
  }

  // routing
  http.Handle("/", Auth(http.HandlerFunc(GetMovie)))

  // jalankan server
  fmt.Println("server running at http://localhost:8080")
  server.ListenAndServe()
}
```

Hasilnya :



Perhatikan kode di atas, untuk mendapatkan username dan password dari auth dengan tipe Basic dapat menggunakan kode `uname, pwd, ok := r.BasicAuth()`, dimana nilai baik tersebut ada 3 yaitu `username(string)`, `password(string)` dan `ok(boolean)`.

Cara kerja nya sama pada contoh sebelumnya, jika username dan password sesuai maka akan di teruskan , namun jika tidak akan di hentikan.

Ingat `method` di atas menggunakan method `POST` ya.

Referensi Tulisan:

Rating - Feedback

Berikan Rating pada posting ini:

Berikan kritik dan saran..

Submit