MODUL PENDAHULUAN KURSUS GOLANG FOR INTERMEDIATE

Versi	1.0
Tahun Penyusunan	2019

Laboratorium Pengembangan Komputerisasi

UNIVERSITAS GUNADARMA

KATA PENGANTAR

Modul ini merupakan modul pendahuluan yang disusun sebagai materi persiapan bagi mahasiswa peserta kursus sebelum setiap pertemuan kursus yang akan dimulai. Materi setiap pertemuan selalu dimulai dengan penjelasan tentang objektif yang akan dicapai dari proses belajar dalam pertemuan tersebut.
Modul GOLANG terdiri dari 8 pertemuan materi :

1. Test Driven Development dan Unit Test
2. Function & Method
3. Structure Query Language (SQL) pada Bahasa Pemrograman Go
4. HTTP REQUEST Pada Bahasa Pemrograman GO
5. Polymer : Properties, Data Binding, Observer & Compute
6. Polymer: Layout Styling 1
7. Polymer: Layout Styling 2
8. Deployment

Secara umum, materi pendahuluan pada setiap pertemuan akan menjelaskan konsep atau teori mengenai topik yang akan dibahas, dan menerangkan secara garis besar langkah yang diperlukan dalam menjalankan suatu software utilitas atau aplikasi yang mendukung pembahasan topik yang dibahas dalam materi tersebut.
Setiap peserta kursus sangat diharapkan untuk mempelajari dengan seksama modul pendahuluan ini, mengingat pemahaman yang baik atas materi ini, akan sangat membantu pada waktu proses belajar selama kegiatan kursus berlangsung, terutama dalam mengerjakan soal-soal Pre-test.
Untuk mengikuti perkembangan teknologi di masa mendatang, maka modul pendahuluan ini disusun oleh Lembaga Pengembangan Komputerisasi Universitas Gunadarma dengan bantuan tim yang bekerja secara penuh yaitu, Drs. Hasin Widjadi, MMSI , Ricky Agus T., ST., SSi., MM , Rheza Andika., SKom., MMSI. , Guntur Eka Saputra, ST., MMSI. , Ahmad Hidayat, SKom., MMSI. , Octaviani Hutapea, ST. , Aria Samudera Elhamidy, SKom. , Azman Agung Nugraha , Ilham Muhammad.

Jakarta, Oktober 2019

Lembaga Pengembangan Komputerisasi

Universitas Gunadarma

DAFTAR ISI

1 Test Driven Development dan Unit Test

Objektif :

- **Mahasiswa diharapkan mampu memahami konsep Test Driven Development**
- **Mahasiswa diharapkan dapat menerapkan konsep Test Driven Development dan Unit Test dalam pembuatan program**

1.1 Pendahuluan

Mengapa sebuah program harus melewati proses pengujian? Terdapat sebuah studi yang dilakukan oleh National Institute of Standart and Technologi (NIST) pada tahun 2002 yang berisikan bahwa bugs pada perangkat lunak menyebabkan kerugian ekonomi di Amerika Serikat sebesar $59.5 billion tiap tahunnya pada saat itu, menurut penelitian tersebut sepertiga dari kerugian ini bisa dihindari jika dilakukan software testing yang lebih baik.

Tujuan utama dari pengujian perangkat lunak sebenarnya sederhana yaitu untuk memastikan bahwa software yang dihasilkan sesuai dengan kebutuhan (requirement) yang sebelumnya ditentukan.

Di bawah ini merupakan beberapa alasan mengapa sebuah program harus melalui tahap pengujian:

1. Untuk menemukan apakah masih terdapat kesalahan atau kekurangan pada program
2. Untuk menyempurnakan program yang baru dibuat jika masih ada kekurangan
3. Untuk membuktikan apakah program tersebut sudah sesuai outputnya seperti yang diinginkan user
4. Untuk melihat performa dari program tersebut
5. Dan hal lain yang menjadi penyebab diperlukanya pengujian pada program

1.2 Unit Test

Unit testing adalah metode verifikasi dan validasi dimana programmer melakukan pengujian terhadap setiap unit pada source code. Sebuah unit merupakan bagian terkecil dari aplikasi yang dapat di uji coba. Unit adalah fungsi individual atau prosedur. Unit test biasanya

dibuat oleh developer dan akan berbeda script dengan tester sendiri. dimana tester memiliki sebuah test sendiri yang biasanya menangani test tentang Functional end point seperti interaksi pengguna dengan aplikasi, API testing, dan lain-lain. Pada Bahasa pemrograman Go disediakan package testing, yang berisikan banyak tools yang dapat digunakan untuk keperluan unit testing.

- TESTING

  Pertama yang harus dibuat merupakan file yang akan di test conoth disimpan dengan nama file latihan.go contoh file berisikan perhitungan luas dan keliling Persegi sederhana.

```
package main

import "math"

type Persegi struct {
    Sisi float64
}

func (k Persegi) Luas() float64 {
    return k.Sisi * k.Sisi
}

func (k Persegi) Keliling() float64 {
    return k.Sisi * 4
}
```

File untuk keperluan testing dipisah dengan file utama, namanya harus berakhiran _test.go, ***namafileutama_test.go*** dan package-nya harus sama. Pada bab ini, file utama adalah latihan.go, maka file testing harus bernama latihan_test.go.

Unit test di Golang dituliskan dalam bentuk fungsi, yang memiliki parameter yang bertipe *testing.T, dengan nama fungsi harus diawali kata Test (pastikan sudah meng-import package testing sebelumnya). Dengan menggunakan parameter tersebut, bisa mengakses method-method untuk keperluan testing. Pada contoh di bawah ini disiapkan 3 buah fungsi test, yang masing-masing digunakan untuk mengecek apakah hasil kalkulasi luas dan keliling Persegi adalah benar.

```
package main
import "testing"
var (
    Persegi              Persegi   = Persegi{5}
    luasSeharusnya     float64 = 25
    kelilingSeharusnya float64 = 20
)
```

```
func TestHitungLuas(t *testing.T) {
    t.Logf("Luas : %.2f", Persegi.Luas())
    if Persegi.Luas() != luasSeharusnya {
        t.Errorf("SALAH! harusnya %.2f", luasSeharusnya)
    }
}
func TestHitungKeliling(t *testing.T) {
    t.Logf("Keliling : %.2f", Persegi.Keliling())
    if Persegi.Keliling() != kelilingSeharusnya {
        t.Errorf("SALAH! harusnya %.2f", kelilingSeharusnya)
    }
}
```

- **Eksekusi File Testing**

  Proses eksekusi file testing adalah menggunakan command go test. Karena struct yang diuji berada dalam file **latihan.go**, maka pada saat eksekusi test menggunakan go test, nama file **latihan_test.go** dan **latihan.go** perlu dituliskan. Berikut merupakan penulisannya:

```
go test namafileutama.go namafileutama_test.go -v
```

  Contoh:

```
go test latihan.go latihan_test.go -v
```

  -v atau –verbose merupakan digunakan untuk menampilkan seluruh output log pada saat user melakukan pengujian.

- ***Method Testing***

  Berikut merupakan tabel dari method testing:

Method	Kegunaan
Log()	Menampilkan log
Logf()	Menampilkan log menggunakan format
Fail()	Menandakan terjadi Fail() dan proses testing fungsi tetap diteruskan
FailNow()	Menandakan terjadi Fail() dan proses testing fungsi dihentikan
Failed()	Menampilkan laporan fail
Error()	Log() diikuti dengan Fail()
Errorf()	Logf() diikuti dengan Fail()
Fatal()	Log() diikuti dengan failNow()
Fatalf()	Logf() diikuti dengan failNow()
Skip()	Log() diikuti dengan SkipNow()

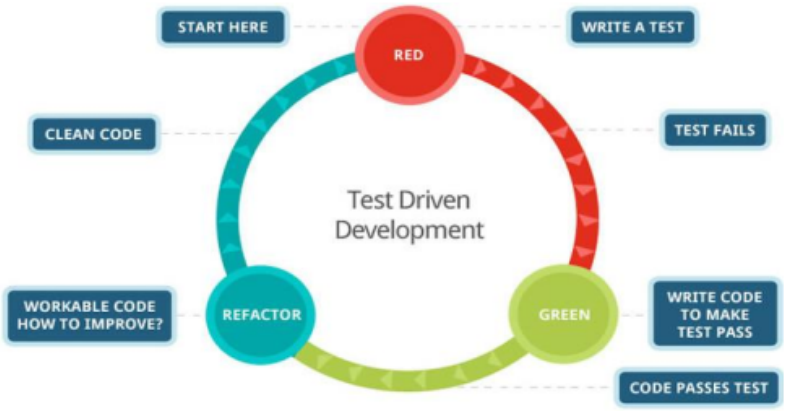
Skipf()	Logf() diikuti dengan SkipNow()
SkipNow()	Menghentikan proses testing fungsi, dilanjutkan ke testing fungsi setelahnya
Skiped()	Menampilkan laporan skip
Parallel()	Menge-set bahwa eksekusi testing adalah parallel

1.3 *Test Driven Development*

- **Definisi**

  Test Driven Development (TDD) adalah teknik pengembangan perangkat lunak yang di mana langkah pengerjaanya adalah membuat kode uji terlebih dahulu sebelum kode produksi dan *refactoring*. Metode TDD mengharuskan pengembang untuk menentukan terlebih dahulu kebutuhan dan desain dari perangkat lunak yang akan dibuat baru kemudian melakukan implementasi kode. Satu poin penting dalam metode TDD adalah pengembangan perangkat lunak fokus pada spesifikasi bukan pada validasinya.

- **Konsep**



Gambar 1.1 Konsep TDD

Pada gambar 1.1 di atas dapat dijelaskan bahwa proses dalam melakukan pengujian terhadap suatu program oleh pengembang menggunakan metode TDD pada awalnya adalah:

1. Sebelum menuliskan kode program, maka tuliskan *test-* nya terlebih dahulu. Pengembang harus memikirkan semua kemungkinan yang dapat terjadi untuk input dan output dari program yang akan dibuat.
2. Jalankan *test-code* tersebuit, dan pastikan *test-code* nya *fail* karena belum ada kode apapun untuk membuat *test-* nya *pass* (sukses) .
3. Ketik *working code* seminimum mungkin dengan tujuan agar *test-* nya *pass*
4. Jalankan *test* dan cek apakah *test-* nya *pass* . Jika belum *pass* , maka perbaiki *working code* sampai memenuhi ekspektasi dari *test* .
5. Jika *working code* yang ditulis sebelumnya menjadi tidak sesuai posisinya. TDD menyediakan fungsi *Refactor the code* , untuk merapihkan kembali kode yang dibuat. Selama *test-* yang dibuat masih *pass,* memiliki arti bahwa kode yang dibuat tidak ada masalah dengan kode yang di *refactor* tersebut.
6. Proses dari no 1–5 dapat diulang untuk fungsi-fungsi lainnya .

- **Kelebihan**

  Berikut merupakan kelebihan dari metode *Test Driven Development*

  1. Pengecekan error lebih mudah dilakukan
  2. Simulasi manual untuk pengujian fungsi perangkat lunak menjadi berkurang
  3. Dengan begitu pengembang dapat melakukan pengcekan lebih cepat
  4. Memudahkan pengguna dalam memodifikasi perubahan
  5. Memudahkan pengembang dalam menemukan *bug*

- **Kekurangan**

  TDD adalah pendekatan yang baik untuk pengembangan perangkat lunak baru tetapi lebih memakan waktu proses model ketika menguji sistem perangkat lunak yang ada.

2 FUNCTION & METHOD

Objektif :

- **Mahasiswa mampu memahami dan menggunakan function pada Golang**
- **Mahasiswa mampu memahami dan menggunakan berbagai function yang sesuai dengan penerapan pada Golang.**
- **Mahasiswa mampu dan menggunakan method pada Golang.**

Pada bagian ini akan membahas tentang fungsi dan method pada Golang yang merupakan aspek penting dalam pemrograman. Definisi fungsi sendiri adalah sekumpulan blok kode yang dibungkus dengan nama tertentu. Setiap program pada Golang memiliki setidaknya satu fungsi, yaitu main ( ). Penerapan fungsi yang tepat akan menjadikan kode lebih modular dan juga *dry* (kependekan dari *don't repeat yourself*), karena tak perlu menuliskan banyak proses berkali-kali, cukup sekali saja dan tinggal panggil jika dibutuhkan. Fungsi juga dikenal sebagai method, sub-routine, atau procedure.

2.1 Penerapan Fungsi

Definisi fungsi dalam bahasa pemrograman Go terdiri dari *function header* dan *function body*. Berikut ini semua bagian dari suatu fungsi:

- **Func** − memulai deklarasi fungsi.
- **Function Name** − nama sebenarnya dari fungsi tersebut. Nama fungsi dan daftar parameter.
- **Parameters** − Parameter seperti tempat penampung. Saat suatu fungsi dipanggil, untuk meneruskan nilai ke parameter. Nilai ini disebut sebagai parameter atau argumen aktual. Daftar parameter mengacu pada jenis, urutan, dan jumlah parameter fungsi. Parameter bersifat opsional; artinya, suatu fungsi mungkin tidak mengandung parameter.
- **Return Type** − Suatu fungsi dapat mengembalikan daftar nilai. Return_types adalah daftar tipe data nilai yang dikembalikan fungsi. Beberapa fungsi melakukan operasi yang diinginkan tanpa mengembalikan nilai. Dalam hal ini, return_type adalah yang tidak diperlukan.

- **Function Body** − Ini berisi kumpulan pernyataan yang menentukan apa fungsi tidak.

Berikut adalah bentuk umum definisi fungsi dalam bahasa pemrograman Golang:

```
func function_name( [parameter list] ) [return_types]
{
   body of the function
}
```

Contoh penerapan perintah function pada Golang:

```
package main

import "fmt"

import "strings"

func main( ) {

var pesan= [] string{"belajar", "Function"}

printMessage( "Yuk", pesan)

}

func printMessage( message string, arr [] string) {

var nameString = strings. Join( arr, " ")

fmt. Println( message, nameString)

}
```

Pada kode di atas dibuat fungsi baru dengan nama ***printMessage*** yang memiliki 2 buah parameter yaitu string ***message*** dan slice string ***arr*** . Di dalam p***rintMessage*** , nilai ***arr*** yang merupakan slice string digabungkan menjadi sebuah string dengan pembatas adalah karakter **spasi**. Penggabungan slice dapat dilakukan dengan memanfaatkan fungsi ***strings***. ***Join( )*** . Fungsi ini berada di dalam package ***strings***.

2.2 Fungsi Dengan Return Value / Nilai Balik

Sebuah fungsi dapat mengembalikan suatu nilai atau dapat didesain tidak mengembalikan apa-apa (void). Fungsi yang memiliki nilai kembalian, harus ditentukan tipe data nilai baliknya pada saat deklarasi. Program berikut merupakan contoh penerapan fungsi yang memiliki return value.

```
package main
import (
"fmt"
"math/rand"
"time"
)
func main( ) {
rand. Seed( time. Now( ) . Unix( ) )
var randomValue int
randomValue = randomWithRange( 2, 10)
fmt. Println( "random number: ", randomValue)
randomValue = randomWithRange( 2, 10)
fmt. Println( "random number: ", randomValue)
```

```
randomValue = randomWithRange( 2, 10)
fmt. Println( "random number: ", randomValue)
}
func randomWithRange( min, max int) int {
var value = rand. Int( ) % ( max - min + 1) + min
return value
}
```

Di dalam fungsi ***randomWithRange*** terdapat proses generate angka acak, yang angka tersebut kemudian digunakan sebagai nilai kembalian.

Cara menentukan tipe data nilai balik fungsi adalah dengan menuliskan tipe data yang diinginkan setelah kurung parameter. Bisa dilihat pada kode di atas, bahwa ***int*** merupakan tipe data nilai balik fungsi ***randomWithRange***.

```
func randomWithRange( min, max int) int
```

Sedangkan cara untuk mengembalikan nilainya adalah dengan menggunakan keyword ***return*** diikuti data yang ingin dikembalikan. Pada contoh di atas, ***return value*** artinya nilai variabel ***value*** dijadikan nilai kembalian fungsi. Eksekusi keyword ***return*** akan menjadikan proses dalam blok fungsi berhenti pada saat itu juga. Semua statement setelah keyword tersebut tidak akan dieksekusi.

Pada kode diatas juga terdapat fungsi ***rand. Seed()*** dimana fungsi ini diperlukan untuk memastikan bahwa angka random yang akan di-generate benar - benar acak. Untuk itu dapat digunakan angka apa saja sebagai nilai parameter fungsi ini (umumnya diisi ***time. Now( ) . Unix( )***).

2.1.1. Deklarasi Perameter Bertipe Sama

Dalam deklarasi ini khusus untuk fungsi yang tipe data parameternya sama, bisa ditulis dengan gaya yang unik. Tipe datanya dituliskan cukup sekali saja di akhir. Contohnya bisa dilihat pada kode berikut.

```
func nameOfFunc( paramA type, paramB type, paramC type) returnType
func nameOfFunc( paramA, paramB, paramC type) returnType
func randomWithRange( min int, max int) int
func randomWithRange( min, max int) int
```

2.3 Fungsi Variadic

Golang mengadopsi konsep variadic function atau pembuatan fungsi dengan parameter sejenis yang tak terbatas. Maksud tak terbatas disini adalah jumlah parameter yang disisipkan ketika pemanggilan fungsi bisa berapa saja. Parameter variadic memiliki sifat yang mirip

dengan slice. Nilai parameter-parameter yang disisipkan memiliki tipe data yang sama, dan akan ditampung oleh sebuah variabel saja. Cara pengaksesan tiap datanya juga sama, dengan menggunakan indeks.

Deklarasi parameter variadic sama dengan cara deklarasi variabel biasa, pembedanya pada parameter jenis ini ditambahkan tanda 3 titik (. . .) setelah penulisan variabel (sebelum tipe data). Nantinya semua nilai yang disisipkan sebagai parameter akan ditampung oleh variabel tersebut.

Berikut merupakan contoh penerapannya:

```
package main
import "fmt"
func main( ) {
var avg = calculate( 2, 4, 3, 5, 4, 3, 3, 5, 5, 3)
var msg = fmt.Sprintf("Rata-rata: %.2f", avg)

fmt.Println( msg)
}
func calculate( numbers ...int) float64 {
var total int = 0
for _, number := range numbers {
total += number
}
var avg = float64( total) / float64( len( numbers) )
return avg
}
```

Fungsi ***calculate( )*** , parameter numbers dideklarasikan dengan disisipkan tanda 3 titik (. . .) sebelum penulisan tipe data-nya. Menandakan bahwa ***numbers*** adalah sebuah parameter variadic dengan tipe data ***int*** .

```
func calculate( numbers . . . int) float64 {
```

Pada pemanggilan fungsi disisipkan banyak parameter sesuai kebutuhan.

```
var avg = calculate( 2, 4, 3, 5, 4, 3, 3, 5, 5, 3)
```

Nilai tiap parameter bisa diakses seperti cara pengaksesan tiap elemen slice. Pada contoh di atas metode yang dipilih adalah ***for - range*** .

```
for _, number := range numbers
```

- Fungsi ***fmt.Sprintf()***

  Fungsi ***fmt.Sprintf( )*** pada dasarnya sama dengan ***fmt.Printf( )***, hanya saja fungsi ini tidak menampilkan nilai, melainkan mengembalikan nilainya dalam bentuk string. Pada

kasus di atas, nilai hasil ***fmt. Sprintf( )*** ditampung oleh variabel ***msg*** . Selain ***fmt.Sprintf( )*** , ada juga ***fmt.Sprint( )*** dan ***fmt.Sprintln( )*** .

- Fungsi ***float64()***

  Sebelumnya sudah dibahas bahwa ***float64*** merupakan tipe data. Tipe data jika ditulis sebagai fungsi (penandanya ada tanda kurungnya) berguna untuk casting. Casting sendiri adalah teknik untuk konversi tipe sebuah data ke tipe lain. Pada contoh di atas, variabel total yang tipenya adalah ***int***, dikonversi menjadi ***float64***, begitu juga **len(numbers)** yang menghasilkan ***int*** dikonversi ke ***float64*** . Variabel ***avg*** perlu dijadikan ***float64*** karena penghitungan rata-rata lebih sering menghasilkan nilai desimal. Operasi bilangan (perkalian, pembagian, dan lainnya) pada Golang hanya bisa dilakukan jika tipe datanya sejenis. Maka dari itulah perlu adanya casting ke tipe ***float64*** pada tiap operand.

- Fungsi Variadic Menggunakan Data Slice

  Slice bisa digunakan sebagai parameter variadic. Caranya cukup mudah, yaitu dengan menambahkan tanda 3 titik setelah nama variabel ketika memasukannya ke parameter. Contohnya bisa dilihat pada kode berikut.

```
var avg = calculate( 2, 4, 3, 5, 4, 3, 3, 5, 5, 3)
var msg = fmt.Sprintf("Rata-rata: %.2f", avg)

fmt.Println( msg)
```

  Pada kode di atas, variabel ***numbers*** yang merupakan slice int, disisipkan ke fungsi ***calculate( )*** sebagai parameter variadic (bisa dilihat tanda 3 titik setelah penulisan variabel). Teknik ini sangat berguna ketika sebuah data slice ingin difungsikan sebagai parameter variadic.

  Perhatikan juga kode berikut ini. Intinya adalah sama, hanya caranya yang berbeda.

```
var numbers = [] int{2, 4, 3, 5, 4, 3, 3, 5, 5, 3}
var avg = calculate( numbers. . . )

// atau

var avg = calculate( 2, 4, 3, 5, 4, 3, 3, 5, 5, 3)
```

  Pada deklarasi parameter fungsi variadic, tanda 3 titik (. . .) dituliskan sebelum tipe data parameter. Sedangkan pada pemanggilan fungsi dengan menyisipkan parameter array, tanda tersebut dituliskan dibelakang variabelnya.

2.4 Fungsi Closure

Definisi termudah Closure adalah sebuah fungsi yang bisa disimpan dalam variabel. Dengan menerapkan konsep tersebut, sangat mungkin untuk membuat fungsi didalam fungsi, atau bahkan membuat fungsi yang mengembalikan fungsi. Closure merupakan *anonymous function* atau fungsi tanpa nama. Biasa dimanfaatkan untuk membungkus suatu proses yang hanya dipakai sekali atau dipakai pada blok tertentu saja.

- **Closure Disimpan Sebagai Variabel**

Sebuah fungsi tanpa nama bisa disimpan dalam variabel. Variabel yang menyimpan closure memiliki sifat seperti fungsi yang disimpannya. Di bawah ini adalah contoh program sederhana untuk mencari nilai terendah dan tertinggi dari suatu array. Logika pencarian dibungkus dalam closure yang ditampung oleh variabel ***getMinMax*** .

```
package main

import "fmt"

func main( ) {
  var getMinMax = func( n [] int) ( int, int) {
    var min, max int
      for i, e := range n {
        switch {
        case i == 0:
         max, min = e, e
        case e > max:
         max = e
        case e < min:
         min = e
        }
     }
     return min, max
  }
  var numbers = [] int{2, 3, 4, 3, 4, 2, 3}
  var min, max = getMinMax( numbers)
  fmt.Printf( "data :%v\nmin :%v\nmax :%v\n", numbers, min, max)
}
```

Pada kode di atas bagiamana cara closure dibuat dan dipanggil. Sedikit berbeda memang dibanding pembuatan fungsi biasa. Fungsi ditulis tanpa nama, lalu ditampung dalam variable.

```
var getMinMax = func( n [] int) ( int, int) {
// . . .
}
```

Cara pemanggilannya, dengan menuliskan nama variabel tersebut sebagai fungsi (seperti pemanggilan fungsi biasa).

```
var min, max = getMinMax( numbers)
```

2.5 Fungsi sebagai Parameter

Pada Golang fungsi bisa dijadikan sebagai tipe data variable dan dapat menjadikannya sebagai parameter juga. Cara membuat parameter fungsi adalah dengan langsung menuliskan skema fungsinya sebagai tipe data. Contohnya bisa dilihat pada kode berikut.

```
package main

import "fmt"
import "strings"

func filter( data [] string, callback func(string) bool) [] string {
var result [] string
for _, each := range data {
if filtered := callback( each) ; filtered {
result = append( result, each)
}
}
return result
}
```

Parameter callback merupakan sebuah closure yang dideklarasikan bertipe ***func(string) bool.*** Closure tersebut dipanggil di tiap perulangan dalam fungsi ***filter( )*** . Fungsi ***filter( )*** sendiri digunakan untuk filtering data array (yang datanya didapat dari parameter pertama), dengan kondisi filter bisa ditentukan sendiri. Di bawah ini adalah contoh pemanfaatan fungsi tersebut.

```
func main( ) {
var data = [] string{"gajah", "buaya", "ular"}
var dataContainsO = filter( data, func(each string) bool {
return strings. Contains( each, "u")
})
var dataLenght5 = filter( data, func(each string) bool {
return len( each) == 5
})
fmt. Println("data asli \t\t: ", data)
// data asli : [gajah buaya ular]
fmt. Println("filter ada huruf \"u\"\t:", dataContainsO)
// filter ada huruf "u" : [gajah]
fmt. Println("filter jumlah huruf \"5\"\t: ", dataLenght5)
// filter jumlah huruf "5" : [gajah buaya]
}
```

Ada cukup banyak hal yang terjadi didalam tiap pemanggilan fungsi ***filter( )*** di atas. Berikut merupakan penjelasannya.

1. Data array (yang didapat dari parameter pertama) akan di-looping
2. Di tiap perulangannya, closure ***callback*** dipanggil, dengan disisipkan data tiap elemen perulangan sebagai parameter
3. Closure ***callback*** berisikan kondisi filtering, dengan hasil bertipe bool yang kemudian dijadikan nilai balik dikembalikan.

4. Di dalam fungsi ***filter( )*** sendiri, ada proses seleksi kondisi (yang nilainya didapat dari hasil eksekusi closure ***callback***). Ketika kondisinya bernilai ***true*** , maka data elemen yang sedang diulang dinyatakan lolos proses filtering.
5. Data yang lolos ditampung variabel ***result*** . Variabel tersebut dijadikan sebagai nilai balik fungsi ***filter( )***

Pada ***dataContainsO*** , parameter kedua fungsi ***filter( )*** berisikan statement untuk deteksi apakah terdapat substring ***"u"*** di dalam nilai variabel ***each*** (yang merupakan data tiap elemen), jika iya, maka kondisi filter bernilai ***true*** , dan sebaliknya. pada contoh ke-2 (***dataLength5***), closure ***callback*** berisikan statement untuk deteksi jumlah karakter tiap elemen. Jika ada elemen yang jumlah karakternya adalah 5, berarti elemen tersebut lolos filter.

2.6 Method

Method adalah fungsi yang hanya bisa di akses lewat variabel objek. Method merupakan bagian dari ***struct*** .Keunggulan method dibanding fungsi biasa adalah memiliki akses ke property struct hingga level *private.* Dan juga, dengan menggunakan method sebuah proses bisa di-enkapsulasi dengan baik. Cara menerapkan method sedikit berbeda dibanding penggunaan fungsi. Ketika deklarasi, ditentukan juga siapa pemilik method tersebut.

Contohnya bisa dilihat pada kode berikut:

```
package main

import "fmt"
import "strings"

type student struct {
name string
grade int

}
func ( s student) sayHello( ) {
fmt.Println( "halo", s.name)
}
func ( s student) getNameAt( i int) string {
return strings. Split( s.name, " ") [i- 1]
}
```

Cara deklarasi method sama seperti fungsi, hanya saja perlu ditambahkan deklarasi variable objek diantara keyword ***func*** dan nama fungsi. Struct yang digunakan akan menjadi pemilik method. ***func (s student) sayHello( )*** maksudnya adalah fungsi ***sayHello*** dideklarasikan sebagai method milik struct ***student*** . Pada contoh di atas struct ***student*** memiliki dua buah method, yaitu ***sayHello( )*** dan ***getNameAt( )*** .

Contoh pemanfaatan method diatas bisa dilihat pada kode berikut:

```
func main( ) {
var s1 = student{"Budi Putra", 17}
s1. sayHello( )
var name = s1.getNameAt( 2)
fmt. Println("nama panggilan : ", name)
}
```

Cara mengakses method sama seperti pengaksesan properti berupa variabel. Tinggal panggil saja methodnya.

```
s1.sayHello( )
var name = s1.getNameAt( 2)
```

Method memiliki sifat yang sama persis dengan fungsi biasa. Seperti bisa berparameter, memiliki nilai balik, dan lainnya. Dari segi sintaks, pembedanya hanya ketika pengaksesan dan deklarasi. Bisa dilihat di kode berikut, sekilas perbandingan penulisan fungsi dan method.

```
func sayHello( ) {
func (s student) sayHello( ) {

func getNameAt(i int) string {
func (s student) getNameAt(i int) string {
```

2.6 Method Pointer

Method pointer adalah method yang variabel objeknya dideklarasikan dalam bentuk pointer. Kelebihan method jenis ini adalah manipulasi data pointer pada property milik variabel tersebut bisa dilakukan.Pemanggilan method pointer sama seperti method biasa.
Contohnya bisa dilihat di kode berikut.

```
func ( s *student) sayHello( ) {
fmt.Println( "halo", s.name)
}

func main( ) {
var s1 = student{"Budi Putra", 17}
s1.sayHello( )
}
```

Method pointer tetap bisa diakses lewat variabel objek biasa (bukan pointer) dengan cara yang nya masih sama. Contoh:

```
// pengaksesan method dari variabel objek biasa
var s1 = student{"Budi Putra", 17}

s1. sayHello( )
// pengaksesan method dari variabel objek pointer
var s2 = &student{"Ari Langit", 21}
s2.sayHello( )
```

3 Structured Query Language (SQL) Pada Bahasa Pemrograman Go

Objektif :

- **Mahasiswa Diharapkan Mampu Mengetahui dan Memahami Konsep SQL**
- **Mahasiswa Diharapkan Mampu Menggunakan Perintah SQL**
- **Mahasiswa Diharapkan Mampu Menggunakan SQL pada Bahasa pemrograman Golang**

3.1 Sekilas Tentang SQL

SQL (*Structured Query Language*) adalah sebuah bahasa yang digunakan untuk berinteraksi dengan database secara terstruktur. Bahasa SQL ini dibuat sebagai bahasa yang dapat merelasikan beberapa tabel dalam database maupun merelasikan antar database. Pada bab ini hanya akan sekilas membahas mengenai sintaks yang ada pada SQL dan akan lebih membahas mengenai cara menggunakan *Database* dengan Bahasa pemrograman Golang.

SQL dibagi menjadi 3 bentuk, yaitu :

1. Data Definition Language (DDL)

   DDL adalah sebuah metode *Query* SQL yang berguna untuk mendefinisikan struktur pada sebuah Database, sintaks yang dimiliki DDL adalah *Create, Drop* dan *Alter*.
2. Data Manipulation Language (DML)

   DML adalah sebuah metode *Query* SQL yang berguna untuk melakukan pemanipulasian data pada *database* yang telah dibuat, DML selalu berhubungan dengan data, sintaks yang dimiliki DML adalah *Select, Insert, Update* dan *Delete*.
3. Data Control Language (DCL)

   DCL adalah sebuah metode *Query* SQL yang digunakan untuk memberikan hak otorisasi mengakses Database, mengalokasikan *space*, pendefinisian *space*, dan pengauditan penggunaan *database*, sintaks yang dimiliki DCL adalah *Grant, Revoke, Commit*, dan *Rollback*.

3.2 Penggunaan SQL pada Bahasa Pemrograman Golang

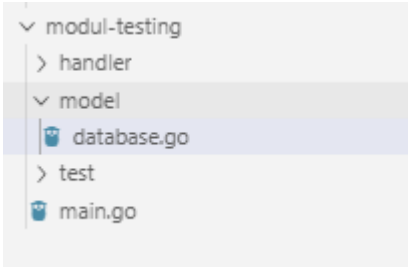
Ketika baru melakukan instalasi golang pada komputer yang digunakan tidak dapat langsung menggunakan query SQL, hal ini dikarenakan *package database/sql* yang berisikan *generic interface* untuk keperluan penggunaan SQL tidak dapat berjalan jika belum tersedia *driver database engine* pada komputer, hal yang harus dilakukan adalah dengan mengunduh *package driver database engine* terlebih dahulu, contoh dari driver *engine* disini adalah Mysql, Oracle, MS SQL Server, dll. Pada modul ini driver yang akan digunakan adalah Mysql. Dibawah ini adalah contoh untuk mengunduh driver mysql.

```
c:\golang\src>go get github.com/go-sql-driver/mysql
```

Gambar 3.1 Mysql Driver

3.3 Data Definition Language (DDL) menggunakan Golang

Pada umumnya pembuatan *database* dan *table* dilakukan langsung menggunakan *mysql console* atau pun *mysql gui* seperti *phpMyAdmin*, namun agar lebih memaksimalkan penggunaan golang, di modul ini pembuatan *database* dan *table* dilakukan melalui kode dan menjalankan *testing* untuk memastikan bahwa *database* dan *table* sudah terbuat. Sebelum memulai pastikan sudah membuat struktur *folder* seperti dibawah ini.



Gambar 3.2 Strukur 1

Setelah itu pada bagian database.go ketikan seperti kode berikut ini

```go
package model

import (
    "database/sql"
    "fmt"
    _ "github.com/go-sql-driver/mysql"
)

type Table interface {
    Name() string
    Field() ([]string, []interface{})
}

func Connect(username, password, host, database string) (*sql.DB, error) {
    conn := fmt.Sprintf("%s:%s@tcp(%s:3306)/%s", username, password, host, database)
    db, err := sql.Open("mysql", conn)
    return db, err
}


func CreateDB(db *sql.DB, name string) error {
    query := fmt.Sprintf("CREATE DATABASE %v", name)
    _, err := db.Exec(query)
    return err
}

func CreateTable(db *sql.DB, query string) error {
    _, err := db.Exec(query)
    return err
}

func DropDB(db *sql.DB, name string) error {
    query := fmt.Sprintf("DROP DATABASE  %v", name)
    _, err := db.Exec(query)
    return err
}
```

Gambar 3.3 Database.GO

- Library

  Pada *file* database.go diatas terdapat 3 library yang digunakan yaitu

  1. database/sql

     Package ini berisikan generic interface untuk keperluan penggunaan SQL.

  2. Fmt

     Package ini mengimplementasikan input atau ouput yang dapat diformat dengan fungsi analog dengan menggunakan printf dan scanf. Penulisan ini mirip seperti pada keluarga Bahasa C namun lebih sederhana.

  3. go-sql-driver/mysql.

     Package ini menyediakan driver MySQL untuk *package database/sq*l. Driver ini harus digunakan bersamaan dengan paket database / sql.

- Fungsi **Connect**

Fungsi ini dibuat untuk melakukan koneksi dengan database, terdapat 4 parameter utama yang harus dimasukan ke dalam fungsi ini, yaitu username, password, database, host. Pada baris ke 15 terdapat penggunaan *fmt.Sprintf* yang berfungsi untuk *formatting* yang akan mengeluarkan hasil string tanpa melakukan cetak apapun, string ini Merupakan *Data Source Name* yang berfungsi untuk melakukan koneksi dengan database, berikut ini adalah bentuk dari koneksi string **(user:pass@tipeprotokol(host:port)/namadb)** sebagai contoh dapat dilihat seperti berikut ini **(root:root@tcp(localhost:3306)/testDatabase),** lalu menggunakan fungsi dari *package database/sql* yaitu **sql.Open** untuk menjalankan koneksi string.

- Fungsi **CreateDB** dan **DropDB**

  Fungsi ini digunakan untuk membuat *database* dan menghapus *database*. Fungsi yang dipakai dari *package database/sql* adalah **db.Exec(query),** fungsi ini digunakan untuk menjalankan query sql yang dimasukan, sebagai contoh dapat dilihat pada baris 21 dan 32 pada Gambar 3.3, baris tersebut adalah baris untuk membuat dan menghapus database, %v yang ada baris tersebut akan digantikan dengan nama database.

3.4 Data Manipulation Language (DML) menggunakan Golang

Pada bagian ini akan mempelajari bagaimana cara menggunakan operasi DML pada Bahasa pemrograman golang, query dml yang digunakan yaitu Select, Insert, Update dan Delete. Sebelum melanjutkan ke dalam program pastikan struktur *folder* sudah seperti dibawah ini.

Gambar 3.4 Struktur 2

Pada modul ini penggunaan Operasi DML di golang akan selalu berhubungan dengan method yang memiliki *method receiver* berupa *struct* dengan *property field-field* yang memiliki nama yang sama dengan *field* dari table di database yang akan digunakan. Pada Gambar 5 terdapat variable TableSiswa yang berisikan suatu string yang akan dimasukan ke

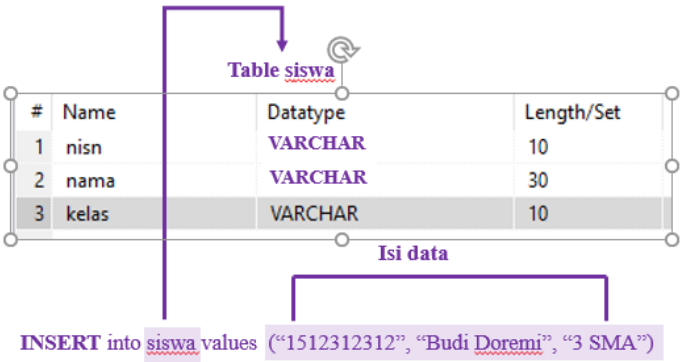
dalam query **db.Exec()** yang terletak pada fungsi **CreateTable** *file* **database.go** untuk membuat suatu tabel, lalu terdapat Struct Siswa yang berisi property berupa field yang ada pada tabel siswa, lalu terdapat method *field* dan *structure* yang berfungsi untuk memberikan pen definiasian dari struct Siswa. Untuk lebih jelasnya dapat dilihat pada Gambar 3.5 dibawah ini.

```go
package model

import (
    "database/sql"
    "fmt"
    "strings"
)

var TabelSiswa string = `
    CREATE TABLE siswa(
        nisn VARCHAR(10) PRIMARY KEY,
        nama VARCHAR(30),
        kelas VARCHAR(10)
    );
`

type Siswa struct {
    NISN  string `json:"NISN"`
    Nama  string `json:"Nama"`
    Kelas string `json:"Kelas"`
}

func (m *Siswa) Fields() ([]string, []interface{}) {
    fields := []string{"nisn", "nama", "kelas"}
    temp := []interface{}{&m.NISN, &m.Nama, &m.Kelas}
    return fields, temp
}

func (m *Siswa) Structur() *Siswa {
    return &Siswa{}
}
```

Gambar 3.5 Siswa.GO

- Fungsi Insert

  Untuk memasukkan data ke dalam sebuah tabel, dapat menggunakan perintah SQL Insert. Berikut adalah anatomi sintaksis untuk perintah *insert* di SQL.



Gambar 3.6 Anatomi Insert

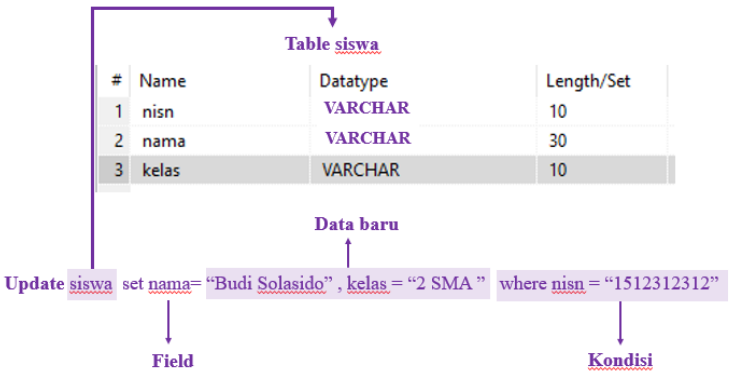
Untuk melakukan operasi *Insert* pada golang, Fungsi yang dipakai dari *package database/sql* adalah **db.Exec(),** seperti pada contoh-contoh sebelumnya, namun dapat dilihat pada baris ke 4, query yang dituliskan berbeda dengan yang ada pada anatomi sintaksis diatas, isi dari values hanya berupa tanda tanya, hal ini berhubungan dengan fungsi **db.Exec()** yang dimana tanda tanya akan digantikan dengan parameter yang ada pada fungsi **db.Exec()**. Untuk lebih jelasnya dapat dilihat pada Gambar 3.7 dibawah ini.

```
33  func (m *Siswa) Insert(db *sql.DB) error {
34      query := fmt.Sprintf("INSERT INTO %v values(?,?, ?)", "siswa")
35      _, err := db.Exec(query, &m.NISN, &m.Nama, &m.Kelas)
36      return err
37  }
38
```

Gambar 3.7 Insert Pada Golang

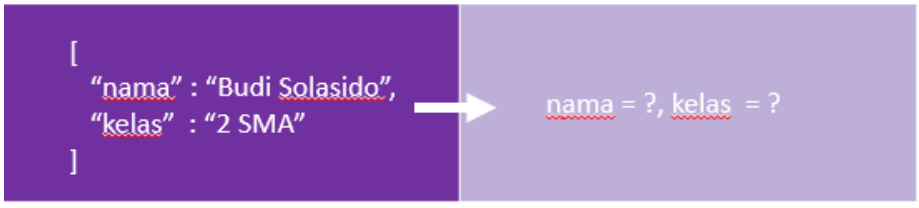
- Fungsi Update

  Untuk melakukan ubah data yang ada pada sebuah table dapat menggunakan perintah SQL Update. Berikut adalah anatomi sintaksis untuk perintah *update* di SQL.



Gambar 3.8 Anatomi Update

Untuk melakukan operasi Update masih menggunakan Fungsi **db.Exec()** seperti pada contoh-contoh sebelumnya, namun terdapat beberapa proses tambahan, seperti pada baris 47-54 yang Merupakan perulangan untuk mengubah format penulisan map menjadi string seperti contoh anatomi dibawah ini.



Gambar 3.9 Convert Data

Tanda tanya diatas akan digantikan dengan isi yang dimasukan ke dalam variable args, lalu pada baris ke 56 kondisi *where* akan dimasukan ke dalam variable args seperti sebelumnya dan akan dieksekusi pada baris ke 58. Untuk lebih jelasnya dapat dilihat pada Gambar 3.10 dibawah ini.

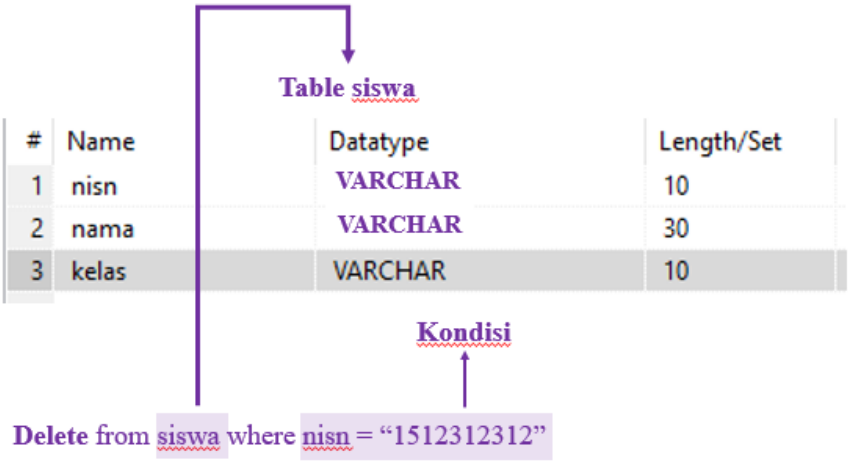
(args… adalah konsep variadic function atau pembuatan fungsi dengan parameter sejenis yang tak terbatas. Maksud tak terbatas disini adalah jumlah parameter yang disisipkan ketika pemanggilan fungsi bisa berapa saja, kalau dalam persamaan matematika seperti args1, args2 …. argsN).

```
39  func (m *Siswa) Update(db *sql.DB, data map[string]interface{}) error {
40
41      var kolom = []string{}
42      var args []interface{}
43
44      i := 1
45
46      // Ini loop data untuk dimasukan kedalam set,
47      for key, value := range data {
48          updateData := fmt.Sprintf("%v = ?", strings.ToLower(key))
49          kolom = append(kolom, updateData)
50          args = append(args, value)
51          i++
52      }
53      // Ubah array menjadi string dengan pemisah koma
54      dataUpdate := strings.Join(kolom, ",")
55      query := fmt.Sprintf("UPDATE %s SET %s WHERE %s = ?", "siswa", dataUpdate, "NISN")
56      args = append(args, m.NISN)
57      // Exec dengan query yang ada
58      _, err := db.Exec(query, args...)
59      return err
60  }
```

Gambar 3.10 Update Pada Golang

- Fungsi Delete

  Untuk melakukan hapus data yang ada pada sebuah tabel dapat menggunakan perintah SQL Delete. Berikut adalah anatomi sintaksis untuk perintah *delete* di SQL.



Gambar 3.11 Anatomi Delete

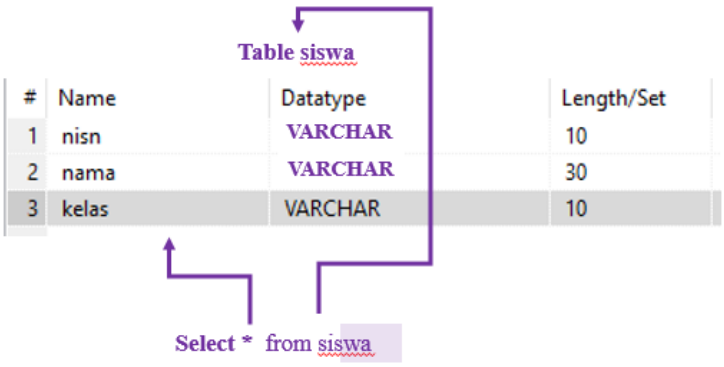
Untuk operasi *Delete* memiliki penggunaan yang sama seperti operasi *Insert*, hanya berbeda pada querynya saja dan parameternya saja yang dimana jika delete parameternya adalah suatu kondisi. Untuk lebih jelasnya dapat dilihat pada Gambar 3.12 dibawah ini.

```
61
62  func (m *Siswa) Delete(db *sql.DB) error {
63      query := fmt.Sprintf("DELETE FROM %s WHERE %s = ?", "siswa", "NISN")
64      // Exec dengan query yang ada
65      _, err := db.Exec(query, m.NISN)
66      return err
67  }
68
```

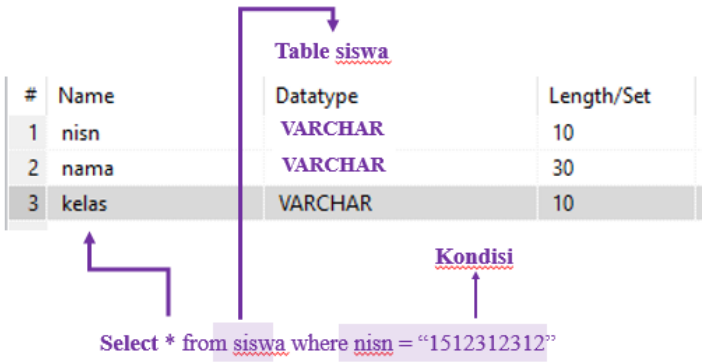
Gambar 3.12 Delete Pada Golang

- Fungsi Get

  Untuk mengambil data pada sebuah tabel dapat menggunakan perintah SQL Select. Dibawah ini terdapat 2 contoh penggunaan Select, yaitu yang mengambil seluruh data dan untuk mengambil 1 data saja berdasarkan kondisi. Berikut adalah anatomi sintaksis untuk perintah *Select* di SQL.



Gambar 3.13 Anatomi Select Seluruh Data



Gambar 3.14 Anatomi Select Satu Data

Untuk operasi Select fungsi yang digunakan adalah **db.Query** dan **db.QueryRow**, perbedaanya adalah dari jumlah data yang akan diambil, jika db.Query akan mengambil seluruh data sedangkan db.QueryRow hanya akan mengambil 1 data saja.

a) Mengambil seluruh data

Untuk mengambil seluruh data, fungsi yang digunakan adalah db.Query, dapat dilihat pada baris ke 87 Gambar 3.15, lalu setelah query dieksekusi diperlukan 1 varibale baru bertipe data pointer Struct untuk menampung data yang akan dimasukan melalui perulangan data.Next(). Perulangan dengan cara ini dilakukan sebanyak jumlah total *record* yang ada, berurutan dari *record* pertama hingga akhir, satu per satu. *Method Scan()* milik *sql.Rows* berfungsi untuk mengambil nilai *record* yang sedang diiterasi, untuk disimpan pada variabel *pointer*. Pada contoh dibawah ini nilai *record* akan dimasukan ke dalam *method Fields* pada *struct* siswa. Untuk lebih jelasnya dapat dilihat pada Gambar 3.15 dibawah ini.

```
84  func GetAllSiswa(db *sql.DB) ([]*Siswa, error) {
85      m := &Siswa{}
86      query := fmt.Sprintf("SELECT * FROM %s", "siswa")
87      data, err := db.Query(query)
88      if err != nil {
89          return nil, err
90      }
91
92      defer data.Close()
93      var result []*Siswa
94      for data.Next() {
95          each := m.Structur()
96          _, dst := each.Fields()
97          err := data.Scan(dst...)
98          if err != nil {
99              return nil, err
100         }
101         result = append(result, each)
102     }
103     return result, nil
104
105 }
```

Gambar 3.15 Select Seluruh Data Pada Golang

b) Mengambil salah satu data

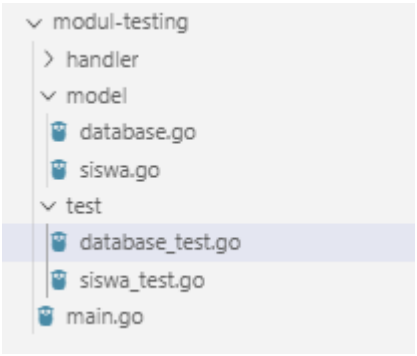
Untuk *query* yang menghasilkan 1 baris *record* saja, bisa gunakan *method db.QueryRow(),* dan bisa langsung di *chaining* atau digabungkan dengan method Scan agar data yang didapatkan akan langsung dimasukan ke dalam *variable each*. Untuk lebih jelasnya dapat dilihat pada gambar dibawah ini.

```go
func Get(db *sql.DB, id string) (*Siswa, error) {

    m := &Siswa{}
    each := m.Structur()
    _, dst := each.Fields()
    query := fmt.Sprintf("SELECT * FROM %s WHERE %s = ?", "siswa", "NISN")

    // isinya akan dimasukan kedalam var dst yang dideklarasikan diatas
    err := db.QueryRow(query, id).Scan(dst...)
    if err != nil {
        return nil, err
    }
    return each, nil
}
```

Gambar 3.16 Select Salah Satu Data Pada Golang

3.5 Membuat File *Testing*

Pada bab sebelumnya sudah dibahas bagaimana caranya melakukan *testing* pada suatu program, pada bagian ini *testing* akan dibuat dan dilakukan pada program yang lebih kompleks dari sebelumnya, yaitu melakukan *testing* pada fungsi-fungsi SQL yang sudah dibuat pada bagian sebelumnya, dimulai dari membuat database, membuat table, mengisi data, melakukan update data, menghapus data, mengambil seluruh data, dan mengambil salah satu data saja. Untuk program *testing*nya dapat dilihat pada kode dibawah ini. Sebelum melanjutkan ke dalam program pastikan struktur *folder* sudah seperti dibawah ini.



Gambar 3.17 Struktur 3

Lalu isikan database_test.go seperti pada Gambar 3.18. Untuk cara menulis *testing* tidak akan dijelaskan kembali dikarenakan sudah dibahas pada bab sebelumnya, untuk isi *testing* sendiri hanya memanggil fungsi SQL yang sebelumnya sudah dibuat dan terdapat 1 fungsi initDatabase yang berguna untuk membuat koneksi secara global.

```go
package test

import (
    "database/sql"
    "fmt"
    "modul-testing/model"
    "testing"
)

var username, password, host, namaDB, defaultDB string

func init() {
    username = "root"
    password = ""
    host = "localhost"
    namaDB = "sekolah"
    defaultDB = "mysql"
}

run test | debug test
func TestDatabase(t *testing.T) {

    t.Run("Testing untuk membuat database", func(t *testing.T) {
        db, err := model.Connect(username, password, host, defaultDB)
        defer db.Close()
        if err != nil {
            t.Fatal(err)
        }
        err = model.CreateDB(db, namaDB)
        if err != nil {
            err = model.DropDB(db, namaDB)
            if err != nil {
                t.Fatal(err)
            }

            err = model.CreateDB(db, namaDB)
            if err != nil {
                t.Fatal(err)
            }
        }

    })

    t.Run("Testing untuk memeriksa koneksi database", func(t *testing.T) {

        db, err := model.Connect(username, password, host, defaultDB)

        defer db.Close()

        if err != nil {
            t.Fatal(err)
        }
    })

```

```
53
54      t.Run("Testing untuk membuat table Siswa", func(t *testing.T) {
55          db, err := model.Connect(username, password, password, namaDB)
56
57          if err != nil {
58              fmt.Println("Gagal melakukan koneksi")
59              t.Fatal(err)
60          }
61
62          if err = model.CreateTable(db, model.TabelSiswa); err != nil {
63              fmt.Println("Gagal membuat table Siswa")
64              t.Fatal(err)
65          }
66      })
67  }
68
69  func initDatabase() (*sql.DB, error) {
70      db, err := model.Connect(username, password, password, namaDB)
71      if err != nil {
72          fmt.Println("Gagal melakukan koneksi")
73          return nil, err
74      }
75      return db, nil
76  }
```

Gambar 3.18 *Testing* DDL

Serta isikan pula siswa_test.go seperti pada Gambar 3.19.

```
1   package test
2
3   import (
4       "modul-testing/model"
5       "testing"
6   )
7
    run test | debug test
8   func TestSiswa(t *testing.T) {
9
10      var dataInsertSiswa = []model.Siswa{
11          model.Siswa{
12              NISN:  "123456731",
13              Nama:  "Budi Doremi",
14              Kelas: "1 SMA",
15          },
```

```
        model.Siswa{
            NISN:  "192837231",
            Nama:  "Doremi Budi",
            Kelas: "2 SMA",
        },
    }

    db, err := initDatabase()
    if err != nil {
        t.Fatal(err)
    }

    t.Run("Testing insert  mahasiswwa", func(t *testing.T) {
        for _, dataInsert := range dataInsertSiswa {
            err := dataInsert.Insert(db)
            if err != nil {
                t.Fatal(err)
            }
        }
    })

    t.Run("Testing update  mahasiswwa", func(t *testing.T) {
        var updateData = map[string]interface{}{
            "Nama": "Susi SUanti",
        }

        data := dataInsertSiswa[0]
        if err := data.Update(db, updateData); err != nil {
            t.Fatal(err)
        }

    })

    t.Run("Testing Get mahasiswa", func(t *testing.T) {
        _, err := model.Get(db, "123456731")
        if err != nil {
            t.Fatal(err)
        }
    })

    t.Run("Testing Get mahasiswa", func(t *testing.T) {
        _, err := model.GetAllSiswa(db)
        if err != nil {
            t.Fatal(err)
        }

    })

    t.Run("Testing delete mahasiswa", func(t *testing.T) {
        data := dataInsertSiswa[1]
        if err := data.Delete(db); err != nil {
            t.Fatal(err)
        }
    })

    defer db.Close()
}
```

Gambar 3.19 *Testing* DML

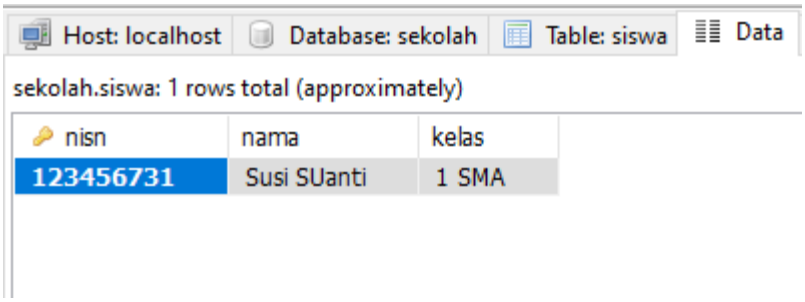
3.6 Melakukan *Testing* pada Fungsi SQL

Untuk melakukan *testing* pastikan terlebih dahulu seluruh *file testing* memiliki akhiran nama *file* berupa _test.go dan pastikan pula *file* berada pada *folder* test. Lalu untuk menjalankan *testing* dapat mengetikan go test -run [Nama Fungsi Yang akan Di *testing*], agar lebih jelasnya dapat dilihat pada Gambar 3.20 dibawah ini.

```
Inspiron        MINGW64 /c/golang/src/modul-testing/test
$ go test -run TestDatabase
PASS
ok      modul-testing/test      0.528s

Inspiron        MINGW64 /c/golang/src/modul-testing/test
$ go test -run TestSiswa
PASS
ok      modul-testing/test      0.564s
```

Gambar 3.20 *Command Testing*

Untuk hasilnya dapat dicek pada Gambar 3.21 dibawah ini, dapat dilihat tabel sudah terbuat dan data sudah di masukan, diupdate dan dihapus.



Gambar 3. 21 Hasil *Testing*

4 *HTTP REQUEST Pada Bahasa Pemrograman GO*

Objektif :

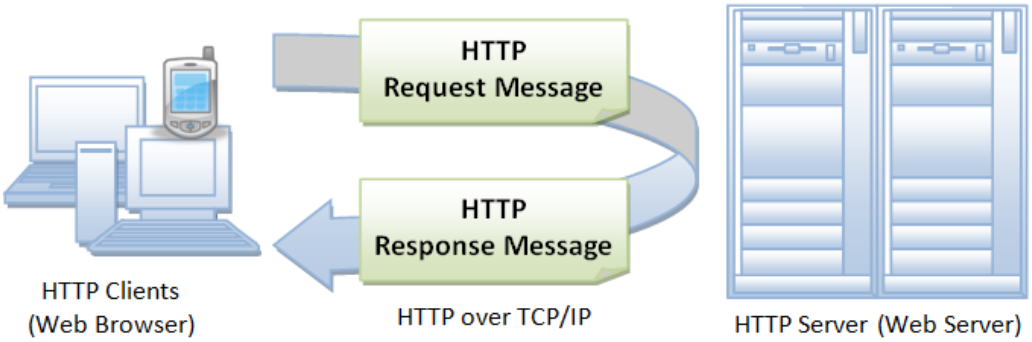
- **Mahasiswa dapat memahami konsep HTTP *Request***
- **Mahasiswa dapat mengenal beberapa metode pada HTTP *Request***
- **Mahasiswa dapat memahami bagaimana implementasi HTTP *Request* pada bahasa pemrograman Golang**

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.

```go
package main

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

type siswa struct {
    ID    string
    Nama  string
    Umur int
}

var data_siswa = []siswa{
    siswa{"A001", "Jojon", 80},
    siswa{"A002", "Riko", 85},
    siswa{"A003", "Bambang", 90},
    siswa{"A004", "Subagja", 70},
}

```

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*.

5 Polymer : Properties, Data Binding, Observer & Compute

Objektif :

- **Mahasiswa dapat memahami tentang Properties Polymer 3**
- **Mahasiswa dapat memahami tentang Data Binding Polymer 3**
- **Mahasiswa dapat memahami tentang Observer Polymer 3**
- **Mahasiswa dapat memahami tentang Compute Polymer 3**

5.1 Apa itu Web Components?

Web components adalah kumpulan fitur pada web browser yang memungkinkan untuk membuat widget (atau yang lebih dikenal dengan istilah component) dalam pengembangan web. Layaknya konsep plugin pada wordpress atau CMS serupa, component ini siap dipakai ulang pada halaman HTML manapun tanpa memerlukan library atau framework tambahan.

Tujuan utamanya adalah memberikan kemudahan untuk penulisan kode dengan struktur berbasis komponen (component-based software engineering). Setiap komponen mewakili satu atau lebih fungsionalitas sistem dan "harus" bersifat reusable. Beberapa framework javascript seperti vue, react atau angular yang sama-sama men-support penulisan kode berbasis komponen juga, namun web components merupakan fitur native di web browser. Contoh sederhananya, misal ada banyak aplikasi web yang dibangun dengan framework yang berbeda-beda. Dengan satu codebase HTML yang sama kita bisa membuat sebuah footer menjadi component, sehingga footer tersebut dapat dipasang di semua aplikasi web menggunakan HTML Imports.

```
<link rel="import" href="my-footer.html">
```

Kemudian memanggilnya dengan Custom Element yang sudah didefinisikan.

```
<my-footer></my-footer>
```

Tidak hanya footer, component bisa saja berupa datepicker, login modal, ads banner, custom button bahkan proses autentifikasi single sign on. Setiap component tidak harus memiliki UI karena sebuah template umumnya terdiri dari blok HTML, CSS dan JS. Bisa saja isinya hanya berupa blok javascript.

Contoh markup elemen HTML Template :

```
<template>
  <style>
    /* BLOCK CSS */
  </style>
  <div>
      <!-- BLOCK HTML -->
  </div>
</template>
<script>
  // BLOCK JS
</script>
```

5.1.1 Apa itu Kerangka (framework) Javascript?

Di dunia development, sebuah "kerangka (framework)", dapat didefinisikan sebagai pustaka Javascript yang menampilkan dan menterjemahkan "data-driven", antarmuka interaktif. Dan digunakan untuk membantu menterjemahkan data ke pengguna saat interaksi dipicu. Semua pustaka (library) sedikit berbeda antara satu dengan yang lainnya, tapi tujuan mereka tetap sama, yaitu menampilkan data baru saat "interaksi" terjadi.

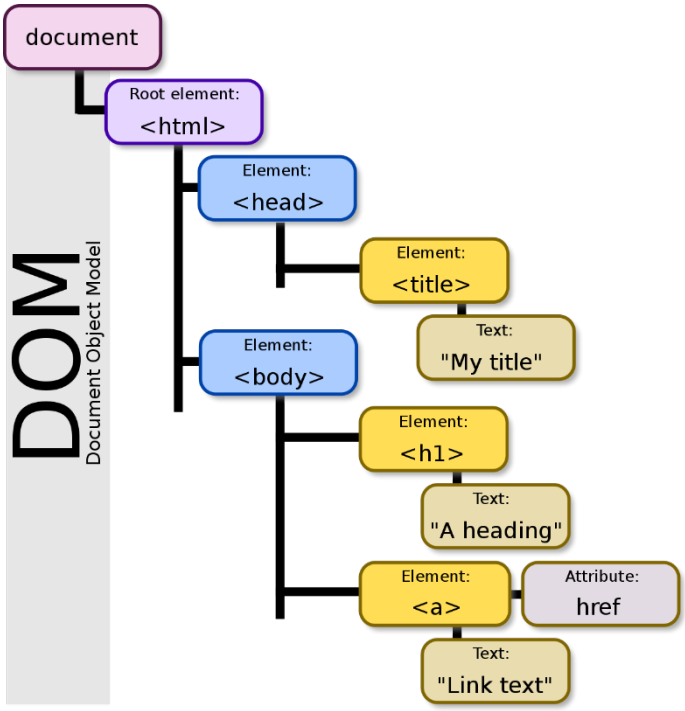
- Pemain-Pemain Kerangka (framework) Javascript

  Ada banyak pilihan terkait memilih kerangka, berikut ini adalah daftar yang ada, termasuk Vue, React, Angular, Ember, polymer dll. Masing masing dipilih berdasarkan perkembangan komunitas dan keberlanjutannya, aktifitas GitHub, seberapa lama proyek ini sudah tersedia untuk developer, dan apakah masih aktif digunakan di group developer dan perusahaan pengembangan.

5.1.2 Polymer JS

Pada bagian modul Google Go language beginner sudah dijelaskan mengenai pengenalan polymer, pada bagian pertemuan modul kali ini akan dibahas mengenai Data Properties, Binding, Observer & Compute pada polymer. Sebelum membahas itu semua, ada baiknya dibahas sedikit mengenai DOM.

DOM (Document Object Model) adalah model data standar. DOM adalah cara javascript melihat suatu halaman html. DOM adalah sebuah platform dan interface yang memperbolehkan pengaksesan dan perubahan pada konten, struktur, dan style pada sebuah dokumen oleh program dan script. Istilah HTML DOM mengacu kepada dokumen html. Kasusnya disini ialah konten, struktur, dan style pada dokumen html dapat diakses dan dirubah dengan menggunakan sintaks javascript.



Gambar 5.1 DOM

DOM pada HTML adalah model objek standar dan antarmuka pemrograman untuk HTML, dan semuanya di definisikan pada :

- Elemen HTML sebagai objek
- Properti semua elemen HTML
- Metode untuk mengakses semua elemen HTML
- Event untuk semua elemen HTML

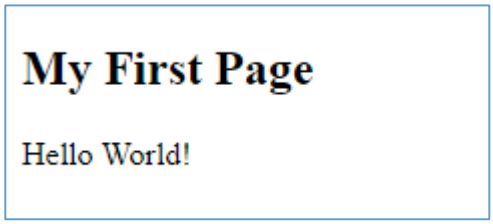
Method DOM HTML adalah tindakan yang dapat Anda lakukan (pada Elemen HTML). Properti DOM HTML adalah nilai (dari Elemen HTML) yang dapat diatur atau ubah. HTML DOM dapat diakses dengan JavaScript (dan dengan bahasa pemrograman lain). Pada DOM semua elemen HTML didefinisikan sebagai objek. Antarmuka pemrograman adalah properti dan metode masing-masing objek. Properti adalah nilai yang bisa Anda dapatkan atau setel (seperti mengubah konten elemen HTML). Method/Metode adalah tindakan yang dapat dilakukan (seperti menambah atau menghapus elemen HTML).

Contoh berikut mengubah konten ***(innerHTML)*** **`<p>`** elemen dengan **`id`**`="demo":`

```
<html>
	<body>
		<p id="demo"></p>
	<script>
```

```
            document.getElementById("demo").innerHTML = "Hello          World!";
      </script>
      </body>
</html>
```

Maka ketika di jalankan, output program tersebut adalah



Dalam contoh di atas, *getElementById* adalah sebuah method, sementara *innerHTML* adalah properti. Cara paling umum untuk mengakses elemen HTML adalah dengan menggunakan id elemen tersebut. Pada contoh di atas *getElementById* adalah metode yang digunakan id="demo"untuk menemukan elemen. cara termudah untuk mendapatkan konten suatu elemen dari Properti *innerHTML* adalah dengan menggunakan *innerHTML* properti. *innerHTML* properti ini berguna untuk mendapatkan atau mengganti isi dari elemen HTML. *innerHTML* properti dapat digunakan untuk mendapatkan atau mengubah setiap elemen HTML, termasuk ***<html>*** dan ***<body>***.

5.2 Properties

Properti adalah atribut yang dapat digunakan dalam pembuatan tampilan yang akan dimunculkan pada bagian depan halaman website. Proses mendeklarasikan properti pada elemen adalah untuk menambahkan nilai default dan mengaktifkan berbagai fitur dalam sistem data. Properti yang dideklarasikan dapat menentukan:

- Jenis properti.
- Nilai standar. (Default)
- *Property change observer*. Memanggil metode kapan pun nilai properti berubah.
- *Read-only status*. Mencegah perubahan yang tidak disengaja pada nilai properti.
- *Two-way data binding support*. Menghidupkan suatu peristiwa setiap kali nilai properti berubah.
- *Computed property*. Secara dinamis menghitung nilai berdasarkan properti lainnya.
- *Property reflection to attribute*. Memperbarui nilai atribut yang sesuai ketika nilai properti berubah.

Banyak fitur-fitur ini terintegrasi erat ke dalam sistem data, dan didokumentasikan dalam bagian sistem data. Selain itu, properti yang dideklarasikan dapat dikonfigurasi dari markup menggunakan atribut. Dalam kebanyakan kasus, properti yang merupakan bagian dari API publik elemen Anda harus dideklarasikan di objek properti. Untuk mendeklarasikan properti, tambahkan pengambil properti statis ke dalam kelas elemen. Pengambil harus mengembalikan objek yang berisi deklarasi properti.

Contoh :

```
class XCustom extends PolymerElement {
  static get properties() {
    return {
      user: String,
      isHappy: Boolean,
      count: {
        type: Number, readOnly: true, notify: true
      }
    }
  }
}
customElements.define('x-custom', XCustom);
```

Keterangan:

Kata Kunci	Keterangan
type	Type: constructor Jenis atribut, digunakan untuk deserializing (Mengubah byte stream ke dalam sebuah objek yang dapat digunakan kembali.) dari suatu atribut. Mendukung Polimer jenis deserializing berikut: Boolean, Tanggal, Nomor, String, Array dan Objek. Anda bisa menambahkan dukungan untuk tipe lain dengan mengganti metode _deserializeValue elemen. Default: String
value	Type: boolean, number, string or function. Nilai default untuk properti. Jika nilai adalah fungsi, fungsi dipanggil dan nilai kembali digunakan sebagai nilai default properti. Jika nilai default harus berupa array atau objek unik untuk instance, untuk array atau objek di dalam suatu fungsi. Default: undefined
reflectToAttribute	Type: boolean Di setting ke true untuk menyebabkan atribut terkait diset pada node host ketika nilai properti berubah. Jika nilai properti adalah Boolean, atribut

	dibuat sebagai atribut boolean HTML standar (diset jika benar, tidak diset jika salah). Default: false
readOnly	Jika benar, properti tidak dapat ditetapkan secara langsung oleh penugasan atau pengikatan data. Default: false
notify	Type: boolean Jika benar, properti tersedia untuk data binding dua arah. Selain itu, suatu event, property-name-changed dilepaskan setiap kali properti berubah. Default: false
computed	Type: string Nilai ditafsirkan sebagai nama metode dan daftar argumen. Metode dipanggil untuk menghitung nilai setiap kali salah satu nilai argumen berubah. Properti yang dihitung selalu hanya read-only. Default: tidak ada
observer	Type: string Nilai ditafsirkan sebagai nama metode yang akan dipanggil ketika nilai properti berubah. Default: tidak ada

5.3 Konsep Sistem Data pada Polymer

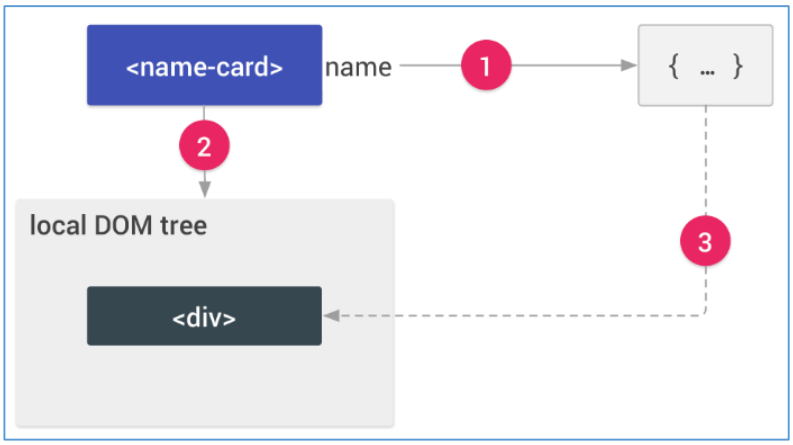
Polymer memungkinkan developer mengamati perubahan pada properti elemen dan mengambil berbagai tindakan berdasarkan perubahan data. Tindakan ini, atau efek propertinya termasuk :

- ***Data bindings***. Anotasi yang memperbarui properti, atribut, atau konten teks dari simpul DOM saat data berubah.
- ***Observers***. Panggilan balik dipanggil saat data berubah.
- ***Computed properties***. Properti virtual dihitung berdasarkan properti lainnya, dan dihitung ulang saat input data berubah.

Setiap elemen Polymer mengelola model datanya sendiri dan elemen DOM lokal. Model untuk elemen adalah properti elemen. Binding data menghubungkan model elemen dengan elemen-elemen di DOM lokalnya.

Contoh :

```
class NameCard extends PolymerElement {
  constructor() {
    super();
    this.name = {first: 'Kai', last: 'Li'};
  }
  static get template() {
    return html `
      <div>[[name.first]] [[name.last]]</div>
    `;
  }
}
customElements.define('name-card', NameCard);
```



Gambar 5.2 Elemen DOM

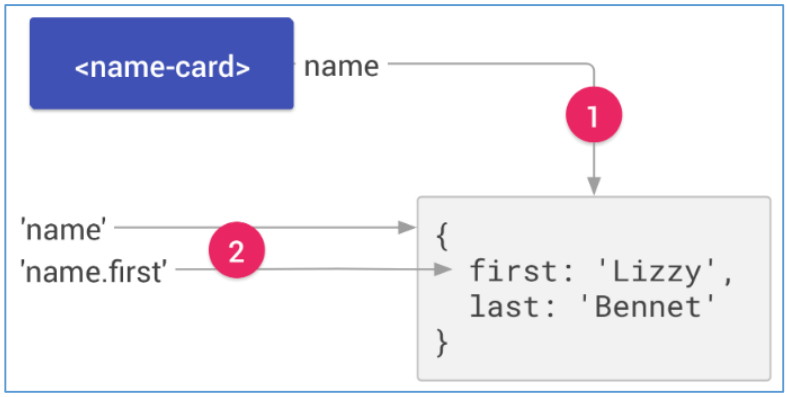
Keterangan :

1. Elemen **<name-card>** memiliki properti nama yang merujuk ke objek JavaScript.
2. Elemen **<name-card>** meng-host tree DOM lokal, yang berisi elemen ** ** tunggal.
3. Binding data dalam templat menautkan objek JavaScript ke elemen ** .**

Sistem data didasarkan pada path, bukan objek, di mana path mewakili properti atau subproperti relatif terhadap elemen host. Sebagai contoh, elemen **<name-card>** memiliki binding data untuk path "name.first" dan "name.last". Jika **<nama-card>** memiliki objek berikut dengan properti namanya:

```
{
  first: 'Lizzy',
  last: 'Bennet'
}
```

Path "name" merujuk ke properti nama elemen (objek). Path "name.first" dan "name.last" merujuk ke properti objek itu.



Gambar 5.3 Path

Keterangan :

1. Properti nama merujuk ke objek JavaScript.
2. Path "name" mengacu pada objek itu sendiri. Path "name.first" mengacu pada subproperti miliknya, pertama (string, Lizzy).

Polymer tidak secara otomatis mendeteksi semua perubahan data.

5.4 Data Binding

Data binding adalah pengikat data dari elemen yang dimodifikasi (element host) ke properti atau atribut elemen di dalam DOM lokal (elemen target). Data element host dapat berupa properti atau sub properti yang diwakili oleh jalur data atau data yang dihasilkan berdasarkan satu jalur atau lebih. Untuk membuat Data binding dengan menambahkan anotasi ke templat DOM lokal elemen, seperti skrip dibawah ini :

```
static get template() {
  return html`
    <target-element          target-property="{{hostProperty}}"></target-
element>`;
}
```

5.4.1 Anatomi Data Binding

Data binding muncul di templat DOM lokal sebagai atribut HTML, seperti skrip dibawah ini :

```
property-name = annotation-or-compound-binding
attribute-name$ = annotation-or-compound-binding
```

Sisi kiri dari binding, mengidentifikasi properti atau atribut target, agar dapat mengikat ke dalam properti, gunakan nama properti dalam bentuk atribut dash-case seperti **`my-element`** untuk HTML dan bukan camelCase untuk di Polymer seperti **`myProperty`** :

```
<my-element my-property = "{{hostProperty}}" ">
```

Contoh binding diatas mengarah ke properti target, **`my-property`** di **`<my-element>.`** Untuk mengikat atribut, dapat menggunakan nama atribut yang diikuti oleh simbol $:

```
<a href$="{{hostProperty}}">
```

Sisi kanan data binding terdiri dari *binding annotation* atau *compound (gabungan) binding*.

- **Binding annotation** adalah Teks yang dikelilingi oleh pembatas kurung kurawal ganda ({{…}}) atau pembatas kurung siku ganda ([[…]]). Mengidentifikasi data host yang sedang terikat.
- **Compound binding** yaitu sebuah string literal yang mengandung satu atau lebih penjelasan yang mengikat.

Apakah aliran data turun dari host ke target, naik dari target ke host, atau kedua cara dikendalikan oleh jenis anotasi yang mengikat dan konfigurasi properti target. Kurung kurawal ganda ({{…}}) mendukung aliran data ke atas dan ke bawah, sedangkan kurung siku kotak ganda ([[…]]) adalah satu arah, dan hanya mendukung aliran data ke bawah.

5.4.2 Binding ke target properti

Untuk mengikat ke properti target, tentukan nama atribut yang sesuai dengan properti, dengan anotasi binding atau compound binding sebagai nilai atribut.

```
<target-element name="{{myName}}"></target-element>
```

Pada contoh skrip diatas mengikat properti nama elemen target ke properti **`myName`** elemen host. Karena anotasi ini menggunakan pembatas dua arah atau "otomatis" ({{…}}),

untuk membuat pengikatan dua arah jika properti nama dikonfigurasikan untuk mendukungnya.

Untuk memastikan binding satu arah, gunakan tanda kurung ganda:

```
<target-element name = "[[myName]]"> </target-element>
```

Untuk mengikat properti elemen camelCase, gunakan dash-case dalam nama atribut. Sebagai contoh:

```
<!-- Bind <user-view>.firstName to this.managerName; -->
<user-view first-name="{{managerName}}"></user-view>
```

Jika properti yang diikat adalah objek atau array, kedua elemen mendapatkan referensi ke objek yang sama. Ini berarti bahwa salah satu elemen dapat mengubah objek dan pengikatan satu arah yang benar adalah hal tidak mungkin.

5.4.3 Bind ke sub-properti host

Data binding juga dapat menyertakan jalur ke sub-properti, seperti yang ditunjukkan di bawah ini:

File : main-view.js

```
import { PolymerElement, html } from '@polymer/polymer/polymer-element.js';
import './user-view.js';

class MainView extends PolymerElement {
  static get properties() {
    return {
      user: {
        type: Object,
        value: function () { return { given: "San", family: "Zhang" }; }
      }
    };
  }
  static get template() {
    return html`
      <user-view given="{{user.given}}" family="{{user.family}}"></user-view>
    `;
```

```
  }
}
customElements.define('main-view', MainView);
```

File : user-view.js

```
import { PolymerElement, html } from '@polymer/polymer/polymer-element.js';

class UserView extends PolymerElement {
  static get properties() {
    return {
      given: String,
      family: String
    };
  }
  static get template() {
    return html`
      <div>[[given]] [[family]]</div>
    `;
  }
}
customElements.define('user-view', UserView);
```

File Pemanggilan : index.html

```
<main-view></main-view>
```

Perubahan sub-properti tidak dapat diamati secara otomatis . Jika elemen host melakukan update ke subproperti, maka perlu menggunakan metode set.

```
//  Change a subproperty observably
this.set('name.last', 'Maturin');
```

Jika pengikatannya dua arah dan elemen target memperbarui properti yang terikat, perubahan akan berpengaruh ke atas (main-view.js) secara otomatis.

5.4.4 Binding ke text content

Untuk mengikat ke konten teks elemen target, cukup menyertakan anotasi atau binding yang mengikat di dalam elemen target.

Contoh :

```
import { PolymerElement, html } from '@polymer/polymer/polymer-element.js';

class UserView extends PolymerElement {
  static get properties() {
    return {
      name: String
    };
  }
  static get template() {
    return html`
      <div>[[name]]</div>
    `;
  }
}
customElements.define('user-view', UserView);
```

Cara menggunakan di HTML nya adalah:

```
<!-- usage -->
<user-view name="Samuel"></user-view>
```

Hasil outputnya adalah Samuel.

5.4.5 Binding ke target attribute

Pada sebagian besar kasus, data binding ke elemen lain harus menggunakan binding properti, di mana perubahan disebarkan dengan menetapkan nilai baru ke properti JavaScript pada elemen. Namun, kadang-kadang perlu mengatur atribut pada elemen, sebagai lawan dari properti. Misalnya, ketika penyeleksi atribut digunakan untuk CSS atau untuk interoperabilitas dengan API berbasis atribut, seperti standar Accessible Rich Internet Applications (ARIA) untuk informasi aksesibilitas. Untuk mengikat atribut, tambahkan tanda dolar ($) setelah nama atribut, seperti contoh dibawah ini :

```
<div style$="color: {{myColor}};">
```

Di mana nilai atribut adalah binding annotation atau sebuah compound binding. Atribut mengikat hasil dalam panggilan ke:

```
element.setAttribute (attr, value);
```

Sebagai gantinya :

```
element.property = value;
```

Contohnya :

```
static get template() {
  return html`
    <!-- Attribute binding -->
    <my-element selected$="[[value]]"></my-element>
    <!-- results in <my-element>.setAttribute('selected', this.value); --
>


    <!-- Property binding -->
    <my-element selected="{{value}}"></my-element>
    <!-- results in <my-element>.selected = this.value; -->
  `;
}
```

Binding atribut selalu satu arah, host-to-target dan nilai diserialisasi menurut jenis nilai saat ini. Ada beberapa properti elemen asli yang umum yang tidak dapat diikat oleh Polymer secara langsung, karena pengikatan menyebabkan masalah pada satu atau lebih browser.

Tabel 5.1 Atribut binding yang mempengaruhi properti

Attribute	**Property**	**Notes**
class	classList, className	Maps to two properties with different formats.
style	style	By specification, style is considered a read-only reference to a CSSStyleDeclaration object.
href	href	-
for	htmlFor	-
data-*	dataset	Custom data attributes (attribute names starting with data-) are stored on the dataset property.
value	value	Only for <input type="number">.

Daftar ini mencakup properti yang saat ini diketahui menyebabkan masalah dengan binding properti. Properti lainnya juga dapat terpengaruh.

Terdapat berbagai alasan mengapa properti tidak dapat diikat:

- Cross-browser mengeluarkan dengan kemampuan untuk menempatkan kurung kurawal {{...}} di beberapa nilai atribut ini.
- Atribut yang memetakan ke properti JavaScript yang diberi nama berbeda (seperti kelas).
- Properti dengan struktur unik (seperti style).

Atribut binding berikut ini adalah yang terikat ke nilai dinamis (gunakan $ =):

```
<!-- class -->
<div class$="[[foo]]"></div>
<!-- style -->
<div style$="[[background]]"></div>
<!-- href -->
<a href$="[[url]]">
<!-- label for -->
<label for$="[[bar]]"></label>
<!-- dataset -->
<div data-bar$="[[baz]]"></div>
<!-- ARIA -->
<button aria-label$="[[buttonLabel]]"></button>
```

5.4.6 Binding ke Array item

Untuk menggunakan computed binding untuk melakukan binding item array tertentu, atau ke subproperti item array, seperti **`array[index].name`**.

Contoh berikut menunjukkan cara mengakses properti dari item array menggunakan computed binding. Function computed perlu dipanggil jika nilai subproperti berubah, atau jika array itu sendiri dimutasi/diganti, sehingga binding akan menggunakan wildcard path, **`myArray.*`**.

```
class XCustom extends PolymerElement {
  static get properties() {
    return {
      myArray: {
        type: Array,
```

```
        value: [{ name: 'Bob' }, { name: 'Wing Sang' }]
      }
    };
  }
  // first argument is the change record for the array change,
  // change.base is the array specified in the binding
  arrayItem(change, index, path) {
    // this.get(path, root) returns a value for a path
    // relative to a root object.
    return this.get(path, change.base[index]);
  }
  ready() {
    super.ready();
    // mutate the array
    this.unshift('myArray', { name: 'Fatma' });
    // change a subproperty
    this.set('myArray.1.name', 'Rupert');
  }
  static get template() {
    return html`
      <div>[[arrayItem(myArray.*, 0, 'name')]]</div>
      <div>[[arrayItem(myArray.*, 1, 'name')]]</div>
    `;
  }
}
```

5.4.7 Binding Dua Arah

Binding dua arah dapat menyebarkan perubahan data baik ke bawah (dari host ke target) dan ke atas (dari target ke host). Agar perubahan menyebar ke atas, Anda harus menggunakan pembatas pengikatan data otomatis ({{…}}) dan properti target harus diset notify: true.

Saat mengikat properti array atau objek, kedua elemen memiliki akses ke array atau objek bersama, dan keduanya dapat membuat perubahan. Gunakan pembatas yang mengikat otomatis untuk memastikan bahwa efek properti merambat ke atas.

5.4.8 Binding Dua Arah non-Polymer elemen

Untuk binding dua arah ke elemen asli atau elemen non-Polimer yang tidak mengikuti konvensi penamaan event, untuk dapat menentukan nama event yang diubah dalam anotasi menggunakan sintaks berikut:

```
target-prop="{{hostProp::target-change-event}}"
```

Contoh :

```
<!-- Listens for `input` event and sets hostValue to <input>.value -->
<input value="{{hostValue::input}}">

<!-- Listens for `change` event and sets hostChecked to <input>.checked -
->
<input type="checkbox" checked="{{hostChecked::change}}">

<!-- Listens for `timeupdate ` event and sets hostTime to
<video>.currentTime -->
<video url="..." current-time="{{hostTime::timeupdate}}">
```

Saat binding ke properti notifikasi standar pada elemen Polimer, menentukan nama event tidak diperlukan, karena konvensi standar adalah menangkap event yang diubah properti.

```
<!-- Listens for `value-changed` event -->
<my-element value="{{hostValue::value-changed}}">

<!-- Listens for `value-changed` event using Polymer convention by default
-->
<my-element value="{{hostValue}}">
```

5.5 Observers properties

Observers adalah metode yang digunakan ketika perubahan yang dapat diamati terjadi pada data elemen. Terdapat dua tipe dasar Observers:

- Observers sederhana yaitu mengamati satu properti.
- Observers kompleks dapat mengamati satu atau lebih properti atau jalur.

Untuk menggunakan sintaks yang berbeda untuk mendeklarasikan dua jenis observers ini, tetapi dalam kebanyakan kasus keduanya berfungsi dengan cara yang sama. Setiap observers memiliki satu atau lebih ketergantungan. Metode observers berjalan ketika perubahan yang diamati dibuat untuk ketergantungan.

Properti yang dihitung adalah properti virtual yang didasarkan pada satu atau lebih bagian data elemen. Properti yang dikomputasi dihasilkan oleh fungsi komputasi, observers kompleks yang mengembalikan nilai. Kecuali ditentukan dengan ketentuan lain, catatan tentang observers berlaku untuk observers sederhana, observers kompleks, dan properti yang dihitung.

5.5.1 Observers dan element initialization

Panggilan pertama ke observers ditunda sampai kriteria berikut dipenuhi:

- Elemen ini sepenuhnya dikonfigurasikan (nilai default telah ditetapkan dan binding data disebarkan).
- Setidaknya salah satu dependensi (ketergantungan) didefinisikan (yaitu, tidak memiliki nilai yang tidak terdefinisi).

Setelah panggilan awal, setiap perubahan yang dapat diamati ke dependensi akan menghasilkan panggilan ke observers, bahkan jika nilai baru untuk dependensi tidak ditentukan. Ini memungkinkan elemen untuk menghindari menjalankan observers dalam kasus default.

5.5.2 Observe sebuah properti

Tetapkan observer sederhana dengan menambahkan kunci observer ke deklarasi properti, mengidentifikasi metode pengamat dengan nama.

Contoh :

```
import { PolymerElement } from '@polymer/polymer/polymer-element.js';

class XCustom extends PolymerElement {
  static get properties() {
    return {
      active: {
        type: Boolean,
        // Observer method identified by name
        observer: '_activeChanged'
      }
    }
  }
  // Observer method defined as a class method
  _activeChanged(newValue, oldValue) {
    this.toggleClass('highlight', newValue);
```

```
  }
}
customElements.define('x-custom', XCustom);
```

Method observer biasanya didefinisikan pada kelas itu sendiri, meskipun metode observer juga dapat didefinisikan oleh superclass, subclass, atau mixin kelas, selama metode yang disebutkan ada pada elemen.

5.5.3 Observers Kompleks

Observers kompleks dideklarasikan dalam array observers. Observers yang kompleks dapat memonitor satu atau lebih path. Path ini disebut dependensi observers.

```
static get observers() {
  return [
    // Observer method name, followed by a list of dependencies, in parenthesis
    'userListChanged(users.*, filter)'
  ]
}
```

Setiap dependensi (ketergantungan) mewakili:

- Properti tertentu (misalnya, firstName).
- Subproperti tertentu (misalnya, address.street).
- Mutasi pada array tertentu (misalnya, users.splices).
- Semua perubahan sub-properti dan mutasi array di bawah jalur yang diberikan (misalnya, pengguna. *).

Method observers dipanggil dengan satu argumen untuk setiap ketergantungan. Berikut ini adalah jenis argumen bervariasi tergantung pada jalur yang diamati, yaitu :

- Untuk properti sederhana atau dependensi subproperti, argumennya adalah nilai baru dari properti atau subproperti.
- Untuk mutasi array argumen adalah catatan perubahan yang menjelaskan perubahan.
- Untuk lintasan wildcard, argumen adalah catatan perubahan yang menjelaskan perubahan, termasuk lintasan persis yang berubah.

Perhatikan bahwa argumen mana pun dapat tidak terdefinisi saat observers dipanggil. observers yang kompleks seharusnya hanya bergantung pada dependensi yang menyatakan.

5.5.4 Observe mengubah pada banyak properti

Observers dapat mengubah pada set properti, maka dapat menggunakan array observers. observers ini berbeda dari pengamat properti tunggal dalam beberapa cara, yaitu:

- Observers multi-properti dan properti yang dihitung berjalan sekali pada inisialisasi jika ada dependensi yang ditentukan. Setelah itu, observers pindah setiap kali ada perubahan yang dapat diamati untuk ketergantungan.
- Observers tidak menerima nilai lama sebagai argumen, hanya nilai baru. Hanya observers properti tunggal yang didefinisikan dalam objek properti yang menerima nilai lama dan baru.

Contoh :

```
import { PolymerElement } from '@polymer/polymer/polymer-element.js';

class XCustom extends PolymerElement {
  static get properties() {
    return {
        preload: Boolean,
        src: String,
        size: String
    }
  }

  // Each item of observers array is a method name followed by
  // a comma-separated list of one or more dependencies.
  static get observers() {
    return [
        'updateImage(preload, src, size)'
    ]
  }
  // Each method referenced in observers must be defined in
  // element prototype. The arguments to the method are new value
  // of each dependency, and may be undefined.
  updateImage(preload, src, size) {
    // ... do work using dependent values
  }
}

customElements.define('x-custom', XCustom);
```

5.6 Computed properties

Computed properties adalah properti virtual yang dihitung berdasarkan satu atau beberapa path. Fungsi komputasi untuk computed properties mengikuti aturan yang sama dengan observer yang kompleks, kecuali bahwa ia mengembalikan nilai, yang digunakan sebagai nilai computed properties.

Seperti observer yang kompleks, fungsi komputasi dijalankan sekali pada inisialisasi jika ada dependensi (ketergantungan) yang telah ditentukan. Setelah itu, fungsi berjalan setiap kali ada perubahan observer pada dependensinya.

5.6.1 Menentukan Computed Properties

Polymer mendukung properti virtual yang nilainya dihitung dari properti lain. Untuk mendefinisikan properti yang dikomputasi, tambahkan ke objek properti dengan key mapping yang dikomputasi ke fungsi komputasi:

```
fullName: {
  type: String,
  computed: 'computeFullName(first, last)'
}
```

Fungsi ini disediakan sebagai string dengan dependensi sebagai argumen yang berada dalam tanda kurung. Seperti halnya observer yang kompleks, fungsi komputasi tidak dipanggil sampai setidaknya satu dependensi didefinisikan (! == tidak terdefinisi).

Contoh :

```
import { PolymerElement, html } from '@polymer/polymer/polymer-element.js';

class XCustom extends PolymerElement {
  static get template() {
    return html`
      <p>My name is <span>{{fullName}}</span></p>
    `;
  }
  static get properties() {
    return {
      first: String,
```

```
      last: String,
      fullName: {
        type: String,
        // when `first` or `last` changes `computeFullName` is called once
        // and the value it returns is stored as `fullName`
        computed: 'computeFullName(first, last)'
      }
    }
  }
  computeFullName(first, last) {
    return first + ' ' + last;
  }
}
customElements.define('x-custom', XCustom);
```

Argumen untuk function computed dapat berupa properti sederhana pada elemen, serta salah satu dari tipe argumen yang didukung oleh observer, termasuk path, path dengan wildcard, dan path ke susunan array. Argumen yang diterima oleh fungsi komputasi cocok dengan yang dijelaskan dalam bagian contoh di atas.

6 Polymer : Layout Styling 1

Objektif :

- **Mahasiswa dapat memahami tentang Layout Polymer 3**
- **Mahasiswa dapat membuat Layout dengan Polymer 3**

6.1 Sekilas Tentang Polymer

Gambar 6.22 Polymer Logo

Polymer adalah *library* JavaScript open-source untuk membangun aplikasi web menggunakan Komponen Web. Perpustakaan sedang dikembangkan oleh pengembang Google dan kontributor di GitHub. Prinsip-prinsip desain modern diimplementasikan sebagai proyek terpisah menggunakan prinsip-prinsip Google Material Design.

Salah satu fitur paling menarik dari Polymer adalah bahwa hal itu memungkinkan para pengembang untuk membuat versi khusus elemen mereka dengan menggunakan CSS, HTML, dan JavaScript untuk menambahkan fitur dalam elemen.

Proyek-proyek dapat dikelola oleh CLI yang disediakan oleh polymer dan sangat berguna dalam menciptakan Aplikasi yang rumit dengan menggunakan elemen yang sangat ramah dan sederhana. Komponen utama Polymer CLI adalah generator boilerplate, pipeline build, test runner, linter, dan development center.

6.2 Instalasi Polymer

Pastikan untuk menginstal Node.js, npm, Git, Bower dan Polymer CLI. Instalasi dapat mengikuti panduan pengaturan di situs web Polymer. https://polymer-library.polymer-project.org/

- Install Node.js

```
choco install -y nodejs-lts
```

- Install git

  https://git-scm.com/download/win download software git dan cek versinya

```
git --version
```

- Install Polymer CLI

  Buka command prompt dan ketikan perintah dibawah.

```
npm install –g polymer-cli
```

Langkah selanjutnya adalah untuk mencoba menjalankan polymer, ketikan perintah untuk cloning script contoh seperti gambar dibawah :

```
git clone https://github.com/PolymerLabs/polymer-3-first-element.git
```

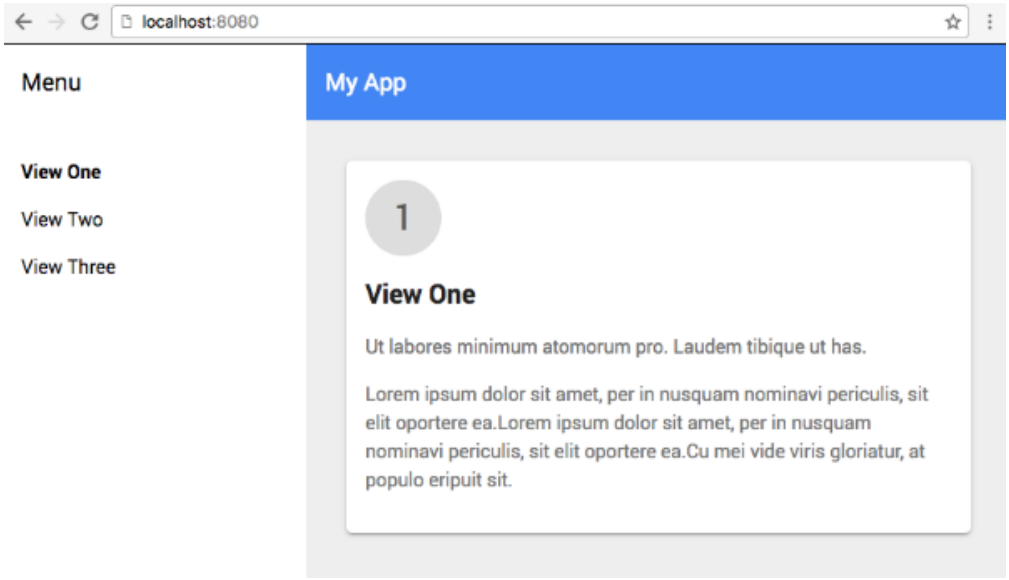
Setelah itu maka akan sampai untuk membangun dependencies

```
cd polymer-3-first-element

npm install
```

jalankan aplikasi dengan perintah :

```
polymer serve
```

Aplikasi dapat di akses di localhost seperti gambar 6.2 dibawah ini.



Gambar 6.23 Tampilan Template Polymer

6.3 Web Komponen

Komponen umumnya dipahami sebagai potongan kode modular yang menyediakan UI dan / atau skrip dalam paket yang dapat digunakan kembali. Banyak kerangka kerja JS menggunakan istilah "komponen" (mis., Angular, React, Ember). Namun, untuk memahami Polymer, kami akan melakukan kursus kilat cepat pada jenis komponen tertentu: komponen web.

Komponen web adalah widget yang dapat digunakan kembali yang dapat dirakit seperti blok bangunan dalam dokumen dan aplikasi web. Mereka adalah serangkaian fitur peramban yang ditambahkan ke spesifikasi W3C HTML dan DOM. Komponen web terdiri dari empat standar:

- Elemen Khusus
- Template HTML
- Bayangan DOM
- Impor HTML

Singkatnya, komponen web memungkinkan kita untuk merancang dan mengimpor elemen khusus yang secara otomatis mengaitkan perilaku JS dengan templat dan dapat memanfaatkan shadow DOM untuk menyediakan pelingkupan CSS dan enkapsulasi DOM.

Komponen web dapat digunakan secara asli tanpa pustaka atau alat tambahan. Namun, tidak semua fitur didukung oleh semua browser. Kita perlu memanfaatkan perpustakaan seperti

Polymer atau polyfill, seperti webcomponents.js, untuk menjembatani kesenjangan antara kondisi dukungan browser saat ini dan masa depan.

6.4 Membangun Elemen Polimer

Hal pertama yang harus dilakukan adalah menginstal beberapa elemen Polymer. Gunakanlah iron-ajax untuk memanggil API dan paper-button untuk UI. Instal komponen ini menggunakan Bower :

```
bower install PolymerElements/iron-ajax PolymerElements/paper-button --save
```

Impor elemen ke dalam home-quotes menggunakan impor HTML. Impor semua dependencies untuk elemen tertentu di file HTML elemen. Agar tidak bergantung pada elemen induk yang memuatnya terlebih dahulu – yang dapat membuat dependencies tidak ada di suatu tempat di kode.

Pustaka polymer dan elemen shared-style sudah diimpor di elemen home-quotes. Tambahkan dua elemen yang baru saja diinstal dengan Bower:

```
<!-- home-quotes.html -->
...
<link rel="import" href="../bower_components/iron-ajax/iron-ajax.html">
<link rel="import" href="../bower_components/paper-button/paper-button.html">
```

6.4.1 Styling

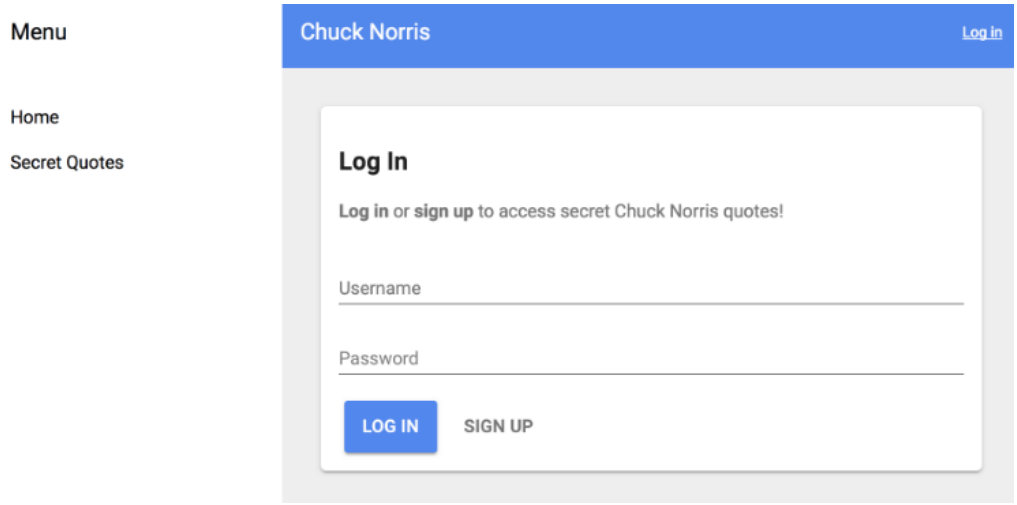
Elemen komponen web menggunakan aturan gaya bayangan DOM. Gaya yang ditentukan dalam root shadow adalah lokal. Ini memungkinkan untuk menargetkan ID dan kelas dalam elemen khusus tanpa merubah halaman atau aplikasi lainnya.

Apabila quotes dan button terlihat agak buruk maka dapat merapikannya dengan CSS. buka file /src/home-quotes.html dan temukan tag <style>. Perhatikan bahwa tag ini menyertakan elemen shared-style. Jangan gunakan gaya blockquote dan paper-button di tempat lain di aplikasi untuk elemen secret-quotes, letakkan style di tempat yang dapat diakses secara global. Kita dapat menggunakan properti CSS khusus dengan Polymer:

```
:root {
    --primary-color: #4285f4;
}
...
a {
    color: var(--primary-color);
}
...
paper-button.primary {
    background: var(--primary-color);
}
```

Banyak elemen Polimer dapat dikustomisasi dengan mengatur variabel. Yang dibuat sendiei. Variable digunakan untuk memberi gaya elemen paper-button dengan kelas .primary.

6.4.2 Mengautentikasi Pengguna di Aplikasi Polymer



Gambar 6.24 Login page

Pengguna harus dapat mendaftar sehingga mereka dapat masuk dan mengakses aplikasi. formulir bagi pengunjung (user) untuk memasukkan kredensial dan data POST ke API untuk mendaftar atau masuk dan menerima token akses JWT. Diperlukan juga penanganan kesalahan mendaftar dan masuk serta menyimpan ke database untuk tetap masuk. gunakan beberapa elemen Polimer dari library, Mulai dengan menginstal iron-input dan paper-input :

```
bower install PolymerElements/iron-input PolymerElements/paper-input
PolymerElements/iron-localstorage -save
```

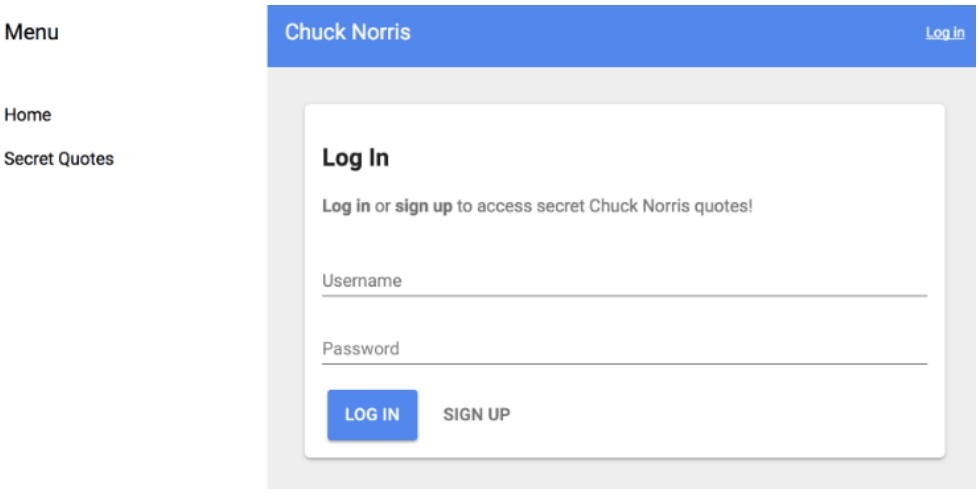
setelah instalasi barulah import library kedalam kode seperti gambar dibawah :

```
<!-- register-login.html -->
<link rel="import" href="../bower_components/polymer/polymer-element.html">
<link rel="import" href="../bower_components/iron-ajax/iron-ajax.html">
<link rel="import" href="../bower_components/iron-localstorage/iron-localstorage.html">
<link rel="import" href="../bower_components/iron-input/iron-input.html">
<link rel="import" href="../bower_components/paper-input/paper-input.html">
<link rel="import" href="../bower_components/paper-button/paper-button.html">
<link rel="import" href="shared-styles.html">
```

Gambar 6.25 Import library

Buatlah tampilan login dengan code sebagai berikut :

```
<div class="card">
    <div id="unauthenticated">
        <h1>Log In</h1>
    <p><strong>Log in</strong> or <strong>sign up</strong> to access secret Chuck Norris quote
    <paper-input-container>
      <label slot="input">Username</label>
      <iron-input slot="input" bind-value="">
        <input id="username" type="text" value="" placeholder="Username">
      </iron-input>
    </paper-input-container>
    <paper-input-container>
      <label>Password</label>
      <iron-input slot="input" bind-value="">
        <input id="password" type="password" value="" placeholder="Password">
      </iron-input>
    </paper-input-container>
    <div class="wrapper-btns">
      <paper-button raised class="primary" on-tap="postLogin">Log In</paper-button>
      <paper-button class="link" on-tap="postRegister">Sign Up</paper-button>
    </div>
</div>
```



Gambar 6.26 Tampilan Login

Hubungkan dengan JS untuk memanfaatkan iron-input dan mengikat nilai ke data yang dapat mengirimkan data ke API. Buat properti yang menyimpan objek untuk data formulir :

```
class RegisterLogin extends Polymer.Element {
  static get is() { return 'register-login'; }

  static get properties() {
    return {
      formData: {
        type: Object,
        value: {}
      }
    }
  }
}
```

Gambar 6.27 Registration code

Properti polimer adalah anggota API publik elemen dan dapat dideklarasikan pada prototipe. Akan digunakan untuk menetapkan nilai default formData ke objek kosong sehingga dapat menambahkan properti hanya dengan mengetikkan bidang input. Menyerahkan Formulir dengan Ajax, gunakan iron-ajax untuk mengirim data formulir kami ke API untuk mendaftar atau login pengguna:

```
<iron-ajax
    id="registerLoginAjax"
    method="post"
    content-type="application/json"
    handle-as="text"
    on-response="handleUserResponse"
    on-error="handleUserError">
</iron-ajax>
```

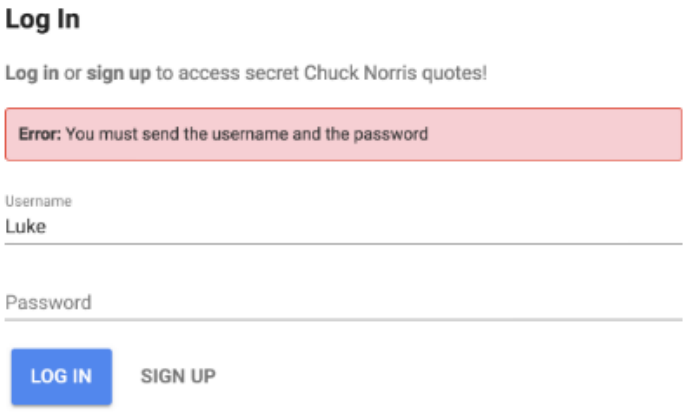
Gambar 6.28 Iron ajax

Gunakan metode POST dan atur tipe konten ke JSON. Menangani respons sebagai teks karena meskipun login yang berhasil mengembalikan JSON, kegagalan mengembalikan string. Dengan cara dapat dengan mudah mem-parsing JSON pada kesuksesan atau menangani teks biasa jika terjadi kesalahan.

Tidak ada atribut url pada elemen iron-ajax saat ini. saatnya menetapkan URL berdasarkan apakah pengguna mengklik "Masuk" atau "Mendaftar". tambahkan atribut ketuk ke dua tombol untuk melampirkan penangan.

```
_setReqBody() {
  this.$.registerLoginAjax.body = this.formData;
}

postLogin() {
  this.$.registerLoginAjax.url = 'http://localhost:3001/sessions/create';
  this._setReqBody();
  this.$.registerLoginAjax.generateRequest();
}

postRegister() {
  this.$.registerLoginAjax.url = 'http://localhost:3001/users';
  this._setReqBody();
  this.$.registerLoginAjax.generateRequest();
}
```

Gambar 6.29 Login



Gambar 6.30 Form Login

7 Polymer : Layout Styling 2

Objektif :

- **Mahasiswa dapat memahami tentang Layout Polymer 3**
- **Mahasiswa dapat membuat Layout dengan Polymer 3**

Setiap aplikasi membutuhkan tata letak (layout), dan elemen tata letak aplikasi menyediakan alat untuk membuat tata letak responsif dengan mudah.

Panel paper-header dan panel paper-drawer, elemen-elemen layout aplikasi. elemen-elemen ini dirancang untuk menjadi:

- Lebih fleksibel dan dapat disusun - mendukung berbagai pola tata letak.
- Kurang berpandangan - elemen-elemen ini tidak memaksakan tampilan dan rasa tertentu (meskipun mereka masih mendukung efek Desain Material dan pola UI jika itu yang Anda cari).
- Dapat dikembangkan - dengan sistem baru yang dapat dicocokkan untuk efek gulir (scroll effect).

Gambar 7.1 Layout

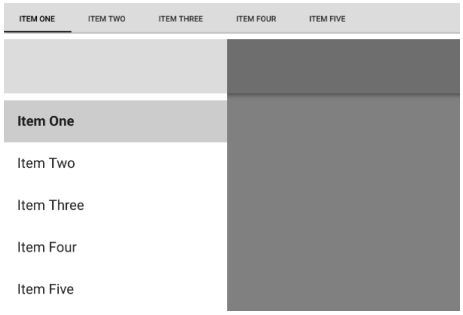
7.1 Rancang tata letak (layout design)

Sebuah layout/tata letak yang baik dibutuhkan dalam sebuah tampilan/user interface yang dapat mendukung jalannya sebuah aplikasi agar sedap dipandang mata.

Ada banyak web komponen yang dapat menunjang dalam mendesign tata letak pada polymer dalam hal ini polymer menyediakan library iron-elements untuk desain layout sederhana dan iron-input-elements yang merupakan elemen untuk desain tata letak input.

7.2 Pola Navigasi Responsif

Dalam banyak kasus, untuk mengganti navigasi ditentukan berdasarkan ukuran layar. Umumnya menggunakan tab navigasi di desktop biasa yang digantikan oleh drawer navigasi di ponsel.



Gambar 7.2 Navigation pattern

Saat ini contoh akan dibuat dalam bentuk input data dalam form input data dan table yang akan digunakan untuk operasi CRUD (Create, Read, Delete, Update) yang didukung oleh elemen dari Polymer

7.2.1 Basic input

berikut adalah basic input yang dapat digunakan dalam polymer. Sebelum dapat menggunakan, lakukanlah instalasi library terlebih dahulu dengan menggunakan perintah :

```
npm install --save @polymer/iron-input
```

lalu masukkan kode sebagai berikut dalam folder html atau element polymer, maka iron-input akan dapat terlihat.

```
<html>
  <head>
    <script type="module">
      import '@polymer/iron-input/iron-input.js';
    </script>
  </head>
```

```
  <body>
    <iron-input>
      <input>
    </iron-input>
  </body>
</html>
```

Atau dalam element polymer

```
import {PolymerElement, html} from '@polymer/polymer';
import '@polymer/iron-input/iron-input.js';
class SampleElement extends PolymerElement {
  static get template() {
    return html`
      <iron-input>
        <input>
      </iron-input>
    `;
  }
}
customElements.define('sample-element', SampleElement);
```

Gambar 7.3 Basic Input Polymer

7.2.2 Iron Form

<iron-form> adalah pembungkus elemen HTML <form>, yang dapat memvalidasi dan mengirimkan elemen HTML kustom dan asli.

Iron-form memiliki dua mode : jika allow-redirect benar, maka setelah pengiriman formulir Anda akan diarahkan ke respons server. Jika tidak, jika itu salah, iron-form akan

menggunakan elemen iron-ajax untuk mengirimkan konten formulir ke server. Langkah pertama yaitu instalasi iron-form :

```
npm install --save @polymer/iron-form
```

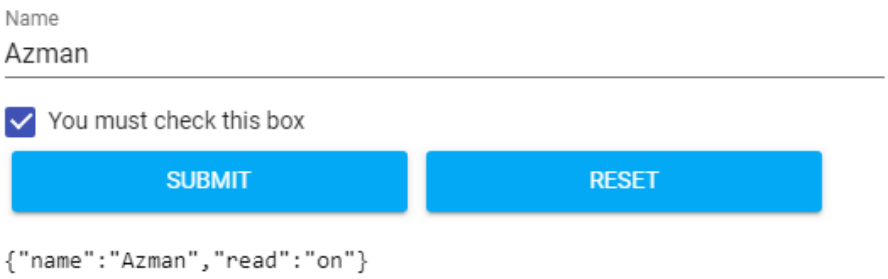
lalu masukkan kode sebagai berikut dalam folder html atau element polymer, maka iron-input akan dapat terlihat.

```
<html>
  <head>
    <script type="module">
      import '@polymer/iron-form/iron-form.js';
      import '@polymer/paper-checkbox/paper-checkbox.js';
    </script>
  </head>
  <body>
    <iron-form>
      <form method="get" action="/form/handler">
        <input type="text" name="name" value="Batman">
        <input type="checkbox" name="donuts" checked> I like
donuts<br>
        <paper-checkbox name="cheese" value="yes" checked></paper-
checkbox>
      </form>
    </iron-form>
  </body>
</html>
```

Atau dalam element polymer

```
import {PolymerElement, html} from '@polymer/polymer';
import '@polymer/iron-form/iron-form.js';
import '@polymer/paper-checkbox/paper-checkbox.js';
class SampleElement extends PolymerElement {
  static get template() {
    return html`
    <iron-form>
      <form method="get" action="/form/handler">
        <input type="text" name="name" value="Batman">
```

```
        <input type="checkbox" name="donuts" checked> I like
donuts<br>
        <paper-checkbox name="cheese" value="yes" checked></paper-
checkbox>
      </form>
    </iron-form>
    `;
  }
}
customElements.define('sample-element', SampleElement);
```



Gambar 7.4 Iron-form

Element pada polymer digunakan dengan berbagai maksud dan tujuan sesuai kegunaannya dan dapat diakses lebih lanjut dalam https://www.webcomponents.org/ yang merupakan katalog dalam library dari Polymer.

7.2.3 Vaadin Grid

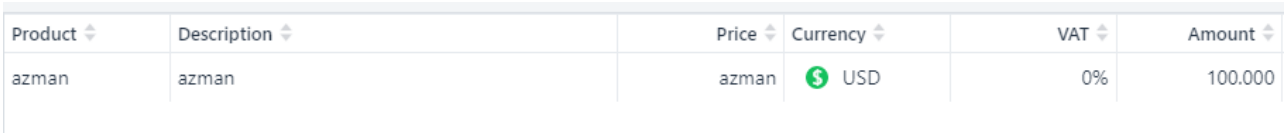
Komponen Vaadin didistribusikan sebagai paket Bower dan npm. Tidak seperti Polymer Elements resmi, komponen Vaadin yang kompatibel dengan Polimer 3 yang dikonversi hanya diterbitkan pada npm. Berikut adalah cara instalasi dari vaadin grid :

```
->  npm i @vaadin/vaadin-grid --save
```

Untuk mengimport vaadin-grid dilakukan pada element polymer yang berekstensi .js dan digunakan untuk berbagai macam table dan berbagai layout yang berhubungan dengan data.

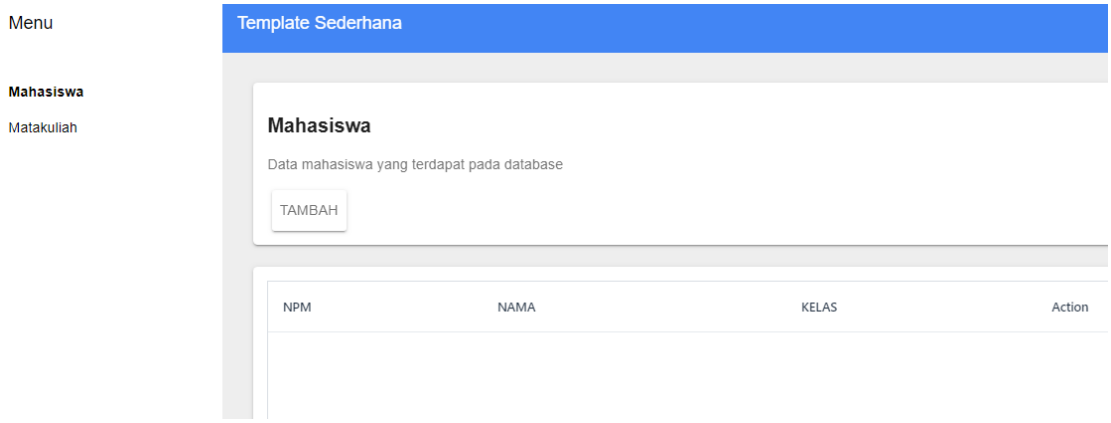
```
import '@vaadin/vaadin-grid/vaadin-grid.js';

<vaadin-grid>
```

```
  <vaadin-grid-column path="name.first" header="First name"></vaadin-grid-column>
  <vaadin-grid-column path="name.last" header="Last name"></vaadin-grid-column>
  <vaadin-grid-column path="location.city"></vaadin-grid-column>
  <vaadin-grid-column path="visitCount" text-align="end" width="120px" flex-grow="0"></vaadin-grid-column>
</vaadin-grid>
```



Gambar 7.5 Vaadin Grid Table

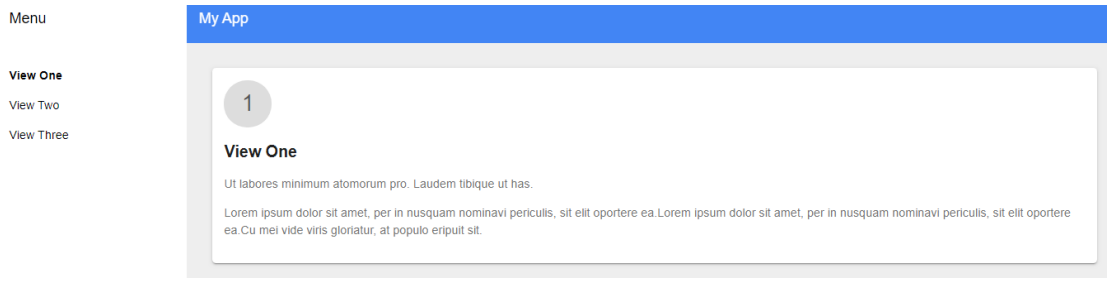
7.3 Template Sederhana



Gambar 7.6 Template Sederhana

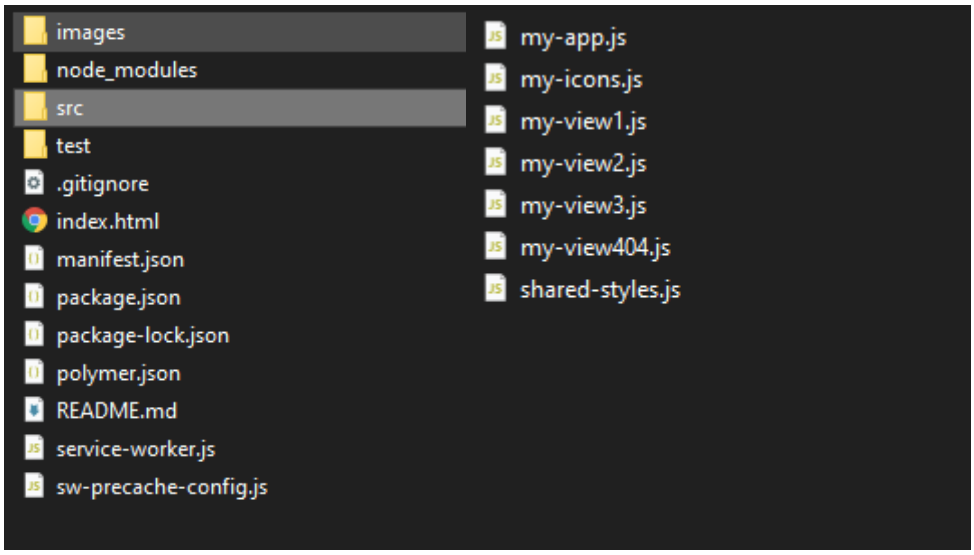
Dalam mempermudah mempelajari dalam pembuatan template dan layout styling, polymer telah menyediakan template sederhana yang dapat kita akses dengan mendownload source yang akan dijalankan pada CLI, berikut adalah bagaimana kita mengambil resource sederhana :

```
-> git clone https://github.com/PolymerLabs/polymer-3-first-element.git
-> npm install
-> polymer serve --open
```



Gambar 7.7 Simple template

Untuk melakukan kostumisasi dan styling pada layout bawaan ini maka gunakan source pada folder src. Melakukan customisasi terdapat beberapa file yang memiliki extension .js dan nama myView. File tersebut merupakan file yg dibangun atas dependencies dari polymer.



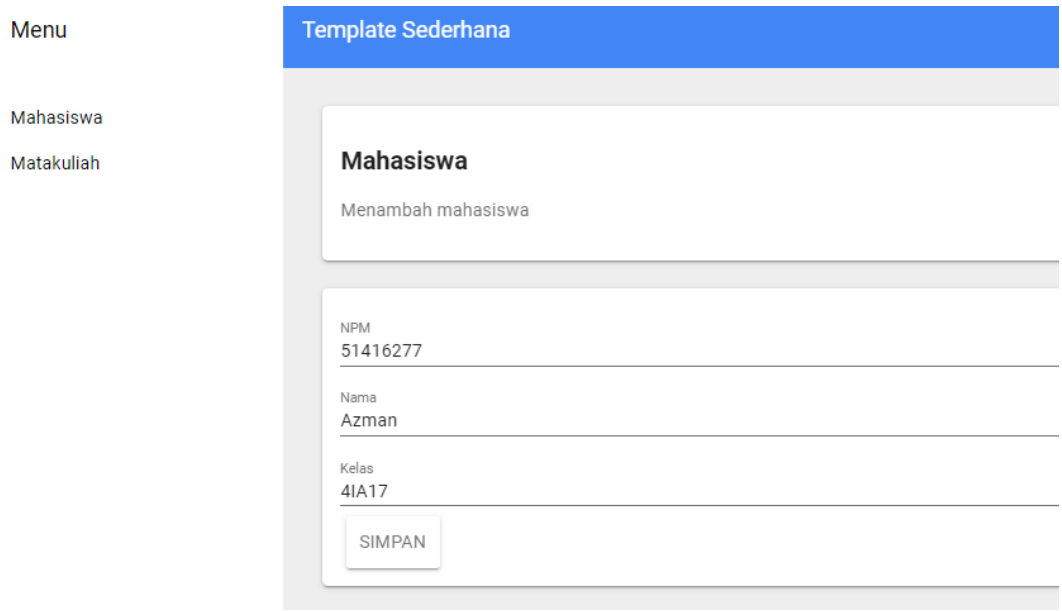
Gambar 7.8 Folder simple template

Sisipkan element polymer seperti basic-input, iron-from,vaadin-grid pada source yang dimiliki oleh my-app serta my-view diatas. Gunakan perintah import untuk menggunakan library yang telah di import.

```
import { PolymerElement, html } from '@polymer/polymer/polymer-
element.js';
import './shared-styles.js';

class MyView1 extends PolymerElement {
  static get template() {
    return html`
      <style include="shared-styles">
        :host {
          display: block;
```

```
            padding: 10px;
          }
        </style>
        <div class="card">
          <div class="circle">1</div>
          <h1>View One</h1>
          <p>Ut labores minimum atomorum pro. Laudem tibique ut has.</
p>
          <p>Lorem ipsum dolor sit amet, per in nusquam nominavi peric
ulis, sit elit oportere ea.Lorem ipsum dolor sit amet, per in nusqua
m nominavi periculis, sit elit oportere ea.Cu mei vide viris gloriat
ur, at populo eripuit sit.</p>
        </div>
      `;
    }
}
window.customElements.define('my-view1', MyView1);
```

setelah semua disisipkan maka tampilan akan dapat di kostumisasi seperti gambar dibawah:



Gambar 7.9 Template input data

8 Deployment

Objektif :

- **Mahasiswa dapat memahami perintah deployment pada Golang**
- **Mahasiswa dapat melakukan *build* program Golang**

8.1 Go Path

Go Path adalah daftar pohon direktori yang berisi kode sumber Go. Ini dikonsultasikan untuk menyelesaikan impor yang tidak dapat ditemukan di Go tree standar. Jalur default adalah nilai dari variabel lingkungan GOPATH, ditafsirkan sebagai daftar Path yang sesuai dengan sistem operasi (pada Unix, variabel adalah string yang dipisahkan titik dua; pada Windows, string yang dipisahkan dengan titik koma). Setiap direktori yang tercantum dalam jalur Go harus memiliki struktur yang ditentukan:

- Direktori src/ menyimpan kode sumber. Jalur di bawah 'src' menentukan jalur impor atau nama yang dapat dieksekusi.
- Direktori pkg/ menyimpan objek paket yang diinstal. Seperti pada pohon Go, setiap sistem operasi target dan pasangan arsitektur memiliki subdirektori pkg sendiri (pkg/GOOS_GOARCH).

Jika DIR adalah direktori yang terdaftar di jalur Go, paket dengan sumber di DIR/src /foo/bar dapat diimpor sebagai "foo/bar" dan formulir kompilasi diinstal ke "DIR/pkg/ GOOS_GOARCH/foo /bar. a "(atau, untuk gccgo," DIR/pkg/gccgo/foo/libbar.a").

bin/ direktori menampung perintah yang dikompilasi. Setiap perintah diberi nama untuk direktori sumbernya, tetapi hanya menggunakan elemen terakhir, bukan seluruh path. Yaitu, perintah dengan sumber di DIR/src/foo/quux diinstal ke DIR/bin/quux, bukan DIR/bin/foo/quux. foo/ dilucuti sehingga Anda dapat menambahkan DIR / bin ke PATH Anda untuk mendapatkan perintah yang diinstal. Contoh berikut adalah tata letak dari folder go path.

```
GOPATH=/home/user/gocode
/home/user/gocode/
    src/
        foo/
            bar/               (kode go pada package bar)
```

```
            x.go
        quux/               (kode go pada package main)
            y.go
bin/
    quux                    (perintah terpasang/installed command)
pkg/
    linux_amd64/
        foo/
            bar.a           (Objek paket yang diinstal)
```

8.2 Building Golang Package for Linux and Windows

Pengembangan aplikasi Go tak jauh dari hal-hal yang menggunakan CLI atau *Command Line Interface*. Proses kompilasi, testing, eksekusi program, semua dilakukan melalui command line. Go menyediakan command `go`.

8.3 Go Build

Command ini digunakan untuk mengkompilasi file program. Ketika eksekusi program menggunakan go run, terjadi proses kompilasi pada blok program, file hasil kompilasi akan disimpan pada folder temporary untuk selanjutnya langsung dieksekusi.

Berbeda dengan go build, command ini menghasilkan file executable pada folder yang sedang aktif. Contohnya bisa dilihat pada kode berikut.

```
PS C:\Users\Rezvan\go\src\activity> go build hello.go
PS C:\Users\Rezvan\go\src\activity> ./hello
Hello Gunadarma
PS C:\Users\Rezvan\go\src\activity>
```

Gambar 8.31 Go Build

Pada contoh diatas file yang di build menghasilkan file baru pada folder yang sama dan dapat dieksekusi. Perintah go build memungkinkan Anda membuat file yang dapat dieksekusi untuk platform target yang didukung Go, Ini berarti unit dapat menguji, merilis, dan mendistribusikan aplikasi tanpa build yang dapat dieksekusi pada platform target yang akan gunakan.

8.4 GOOS and GOARCH

Kompilasi silang bekerja dengan menetapkan variabel lingkungan yang diperlukan yang menentukan sistem operasi dan arsitektur target. Gunakan GOOS variabel untuk sistem operasi target, dan GOARCH untuk arsitektur target. Untuk build yang dapat dieksekusi, perintah akan menggunakan ini :

```
-> env GOOS=target-OS GOARCH=target-architecture go build package-import-path
```

Perintah env menjalankan program dalam lingkungan yang dimodifikasi. Ini memungkinkan untuk menggunakan environment variabel hanya untuk eksekusi perintah saat ini. Variabel tidak disetel atau direset setelah perintah dijalankan. Tabel berikut menunjukkan kemungkinan kombinasi GOOS dan GOARCH yang dapat digunakan:

Tabel 8.1 Perintah GOOS dan GOARCH

GOOS - Target Operating System	GOARCH - Target Platform
android	arm
darwin	386
darwin	amd64
darwin	arm
darwin	arm64
dragonfly	amd64
freebsd	386
freebsd	amd64
freebsd	arm
linux	386
linux	amd64

GOOS - Target Operating System	GOARCH - Target Platform
linux	arm
linux	arm64
linux	ppc64
linux	ppc64le
linux	mips
linux	mipsle
linux	mips64
linux	mips64le
linux	s390x
nacl	386
nacl	amd64p32
nacl	arm
netbsd	386
netbsd	amd64
netbsd	arm
openbsd	386
openbsd	amd64
openbsd	arm
plan9	386

GOOS - Target Operating System	GOARCH - Target Platform
plan9	amd64
plan9	arm
solaris	amd64
windows	386
windows	amd64

Menggunakan nilai-nilai dalam tabel, kita dapat mem-build program Go untuk Windows 64-bit seperti ini:

```
env GOOS=windows GOARCH=amd64 go build main.go
```

Tidak ada output yang menunjukkan bahwa operasi berhasil. Eksekusi akan dibuat di direktori saat ini, menggunakan nama paket sebagai namanya. Untuk membuat perintah dapat dijalankan untuk Windows, nama diakhiri dengan akhiran .exe.

Dalam pembahasan kali ini cara menggunakan perintah Go untuk mendapatkan paket dari sistem kontrol, serta membangun dan mengkompilasi yang dapat dieksekusi untuk platform yang berbeda. Perintah ini juga dapat membuat skrip yang bisa gunakan untuk mengkompilasi silang (Cross-compiling) satu package untuk banyak platform. Untuk memastikan aplikasi berfungsi dengan benar.

Sumber 1

main.go

```
package main

func main() {}
```

make.sh

```
#!/bin/sh

os_archs=()

# Reference:
# https://github.com/golang/go/blob/master/src/go/build/syslist.go
```

```
for goos in android darwin dragonfly freebsd linux nacl netbsd openbsd plan9
solaris windows zos
do
    for goarch in 386 amd64 amd64p32 arm armbe arm64 arm64be ppc64 ppc64le mips
\
        mipsle mips64 mips64le mips64p32 mips64p32le ppc s390 s390x sparc sparc64
    do
        GOOS=${goos} GOARCH=${goarch} go build -o /dev/null main.go >/dev/null
2>&1
        if [ $? -eq 0 ]
        then
            os_archs+=("${goos}/${goarch}")
        fi
    done
done

for os_arch in "${os_archs[@]}"
do
    echo ${os_arch}
done
```

Sumber 2

main.go

```
package main

const (
        hello uint = 0xfedcba9876543210
)

func main() {}
```

make.sh

```
#!/bin/bash

# Reference:
# https://github.com/golang/go/blob/master/src/go/build/syslist.go
os_archs=(
    darwin/386
    darwin/amd64
    dragonfly/amd64
    freebsd/386
    freebsd/amd64
    freebsd/arm
    linux/386
    linux/amd64
    linux/arm
    linux/arm64
    linux/ppc64
    linux/ppc64le
    linux/mips
    linux/mipsle
    linux/mips64
```

```
    linux/mips64le
    linux/s390x
    nacl/386
    nacl/amd64p32
    nacl/arm
    netbsd/386
    netbsd/amd64
    netbsd/arm
    openbsd/386
    openbsd/amd64
    openbsd/arm
    plan9/386
    plan9/amd64
    plan9/arm
    solaris/amd64
    windows/386
    windows/amd64
)

os_archs_32=()
os_archs_64=()

for os_arch in "${os_archs[@]}"
do
    goos=${os_arch%/*}
    goarch=${os_arch#*/}
    GOOS=${goos} GOARCH=${goarch} go build -o /dev/null main.go >/dev/null 2>&1
    if [ $? -eq 0 ]
    then
        os_archs_64+=(${os_arch})
    else
        os_archs_32+=(${os_arch})
    fi
done

echo "32-bit:"
for os_arch in "${os_archs_32[@]}"
do
    printf "\t%s\n" "${os_arch}"
done
echo

echo "64-bit:"
for os_arch in "${os_archs_64[@]}"
do
    printf "\t%s\n" "${os_arch}"
done
echo
```

DAFTAR PUSTAKA

[1] Beck, K. (2003). Test-Driven Development: By Example. Boston : Addison-Wesley.

[2] Bissi, W., Neto, A.G.S.S., Emer, M.C.F.P. 2016. "The Effect of Test Driven Development on Internal Quality, External Quality and Production : A Systematic Review", in Information andSoftware Technology, pp. 2 – 35.

[3] Doxsey, Caleb. 2016. Introducing Go: Build Reliable, Scalable Programs. O'Reilly Media

[4] URL : https://www.webcomponents.org , 12 Agustus 2019

[5] URL: https://kursuswebprogramming.com, 21 Agustus 2019