**Backend Golang Documentation**

**Installing Go**

1. Install go from:

https://golang.org/dl/

**Creating Go project**

Method 1:

1. Create file project1.go
2. Make the code inside project1.go
3. On cmd, type:

go run project1.go

Method 2:

1. On cmd, type:

go mod init project1

There will be file go.mod as a list of modules dependency of project1

1. Create file blablaba.go
2. Make the code inside blablaba.go
3. On cmd, type:

go build

There will be executable file project1.exe based on what name defined in go mod

**Go variables and constant**

Declaring variable:

1. Method 1:

var text1 = "this is text1"

var text1 string = "this is text1"

var positiveNumber uint8 = 89

var message string = `Nama saya "John Wick".

Salam kenal.

Mari belajar "Golang".`

1. Method 2

text1 := "this is text1"

text1 = "updated this is text1" 🡪 use ‘=’ (updating) only after ‘:=’ (declaring)

Declaring constant:

1. const text1 = “contant text”

**Go data type**

1. uint 🡪 unsigned integer, equal to uint32 or uint64, default value 0
2. int 🡪 integer, equal to int32 or int64
3. uint8/byte, uint16, uint32, uint64 🡪 defined bit length
4. int8, int16, int32, int64 🡪 defined bit length for int, e.g. int8 -128 to 127
5. float32, float64 🡪 floating point, default value 0.0
6. bool 🡪 true or false, default value false
7. string 🡪 string, default value “”
8. nil 🡪 empty value, can be assigned to pointer, func data type, slice, map, channel, interface

**Print in Go**

1. Method 1:

fmt.Printf("%d %3.3f %t \n", nonDecimalNum, decimalNum, boolVal)

* %d 🡪 non decimal
* %f 🡪 decimal
* %t 🡪 bool
* \n 🡪 new line
* \t 🡪 tab space

1. Method 2, use backticks:

var name = "John Doe"

var sentence = `halo nama saya ` + name

fmt.Println(sentence)

**String package in Go**

1. strings.index()

var index1 = strings.Index("ethan hunt", "ha")

fmt.Println(index1) // 2

1. strings.Replace()

var text = "banana"

var find = "a"

var replaceWith = "o"

var newText1 = strings.Replace(text, find, replaceWith, 2)

fmt.Println(newText2) // "bonona"

1. strings.Repeat()

var str = strings.Repeat("na", 4)

fmt.Println(str) // "nananana"

1. strings.ToLower()

var str = strings.ToLower("aLAy")

fmt.Println(str) // "alay"

1. strings.ToUpper()

var str = strings.ToUpper("eat!")

fmt.Println(str) // "EAT!"

1. strconv.Atoi() 🡪 string to int

import "strconv"

var str = "124"

var num, err = strconv.Atoi(str)

if err == nil {

fmt.Println(num) // 124

}

1. strconv.Itoa() 🡪 int to string

var num = 124

str, \_ := strconv.Itoa(num)

fmt.Println(str) // "124"

1. strconv.ParseInt() 🡪 str to int

var str = "124"

num, \_ := strconv.ParseInt(str, 10, 64)

fmt.Println(num) // 124

1. strconv.FormatInt() 🡪 returns the string representation of an int in the given base

var num = int64(24)

str := strconv.FormatInt(num, 8)

fmt.Println(str) // 30

1. strconv.ParseFloat() 🡪 str to e.g. float32

var str = "24.12"

num, \_ := strconv.ParseFloat(str, 32)

fmt.Println(num) // 24.1200008392334

1. strconv.FormatFloat() 🡪 float64 to str

var num = float64(24.12)

str := strconv.FormatFloat(num, 'f', 6, 64)

fmt.Println(str) // 24.120000

1. strconv.ParseBool() 🡪 str to bool

var str = "true"

var bul, \_ = strconv.ParseBool(str)

fmt.Println(bul) // true

1. strconv.FormatBool() 🡪 bool to str

var bul = true

var str = strconv.FormatBool(bul)

fmt.Println(str) // true

**Conditional in Go**

1. if-else if-else conditionals:

var minimarketStatus = "close"

var minuteRemainingToOpen = 5

if minimarketStatus == "open"{

fmt.Println("saya akan membeli telur dan buah")

} else if minuteRemainingToOpen <= 5 {

fmt.Println("minimarket buka sebentar lagi, saya tungguin")

} else {

fmt.Println("minimarket tutup, saya pulang lagi")

}

1. if-else if-else conditionals, variable in conditional declared inside if:

if minimarketStatus, minuteRemainingToOpen := "close", 5; minimarketStatus == "open"{

fmt.Println("saya akan membeli telur dan buah")

} else if minuteRemainingToOpen <= 5 {

fmt.Println("minimarket buka sebentar lagi, saya tungguin")

} else {

fmt.Println("minimarket tutup, saya pulang lagi")

}

1. switch case:

var point = 6

switch {

case point == 8:

fmt.Println("perfect")

case (point < 8) && (point > 3):

fmt.Println("awesome")

fallthrough 🡪 make sure that conditional checking continued to the next cases

case point < 5:

fmt.Println("you need to learn more")

default:

{

fmt.Println("not bad")

fmt.Println("you need to learn more")

}

}

**Array in Go**

Same type, predefined length

Initializing array:

1. Method 1, empty array:

var names = []string{}

1. Method 2, predefining the length:

var names [4]string

names[0] = "John"

names[1] = "Doe"

names[2] = "Frank"

names[3] = "Jack"

or

var names = [4]string{"John", "Doe", "Frank", "Jack"}

1. Method 3, without predefined length 🡪 length will be automatically counted

var numbers = [...]int{2, 3, 2, 4, 3}

1. Array inside array:

var numbers1 = [2][3]int{[3]int{3, 2, 3}, [3]int{3, 4, 5}}

var numbers2 = [2][3]int{{3, 2, 3}, {3, 4, 5}} 🡪 inside array can be typed simpler

**Slice in Go**

Reference of array at certain memory space. If the array changes, the slice changes as well. No predefined length.

Initializing slice

1. Method 1:

var fruits = []string{"apple", "grape", "banana", "melon"}

1. Method 2, using 2 index from an array:

var fruits = []string{"apple", "grape", "banana", "melon"}

var newFruits = fruits[0:2]

if only use 1 index, it will make a copy of array’s element

if use 2 index, it will call a reference element

**Function at Array/Slice in Go**

1. len() 🡪 find the length of an array/slice

var fruits = []string{"apple", "grape", "banana", "melon"}

fmt.Println(len(fruits)) // 4

1. cap() 🡪 find the maximum capacity of an array/slice

var fruits = []string{"apple", "grape", "banana", "melon"}

var aFruits = fruits[0:3]

fmt.Println(cap(aFruits)) // cap: 4

var bFruits = fruits[1:4]

fmt.Println(cap(bFruits)) // cap: 3

1. append() 🡪 adding element to an array/slice

var fruits = []string{"apple", "grape", "banana"}

var appenedFruits = append(fruits, "papaya")

1. copy() 🡪 copying value at src to dst, returning the number of values copied

dst := make([]string, 3)

src := []string{"watermelon", "pinnaple", "apple", "orange"}

n := copy(dst, src)

fmt.Println(dst) // watermelon pinnaple apple

fmt.Println(src) // watermelon pinnaple apple orange

fmt.Println(n) // 3

1. Accessing slice element with 3 indexing 🡪 3rd index define the capacity

var fruits = []string{"apple", "grape", "banana"}

var aFruits = fruits[0:2]

var bFruits = fruits[0:2:2]

fmt.Println(cap(aFruits)) // cap: 3

fmt.Println(cap(bFruits)) // cap: 2

1. Allocating silce element with make

var fruits = make([]string, 2)

**Map in Go**

Key-value pair

Initializing map:

1. Method 1:

var myMap map[string]int

myMap = map[string]int{} 🡪 end bracket defines default values

myMap ["januari"] = 50

myMap ["februari"] = 40

fmt.Println("januari", myMap ["januari"]) // januari 50

fmt.Println("mei", myMap ["mei"]) // mei 0

1. Method 2:

myMap := map[string]int{} 🡪 simpler to use than above!

myMap ["januari"] = 50

myMap ["februari"] = 40

fmt.Println("januari", myMap ["januari"]) // januari 50

fmt.Println("mei", myMap ["mei"]) // mei 0

1. Method 3:

var myMap = map[string]int{"januari": 50, "februari": 40}

1. Method 4:

var myMap = map[string]int{

"januari": 50,

"februari": 40,

}

1. Method 5:

var myMap = make(map[string]int)

1. Method 6:

var myMap = \*new(map[string]int)

Adding element to map:

1. myMap["januari"] = 50

Deleting element from map:

1. delete(myMap, "januari")

checking element existing in map:

1. var value, isExist = myMap["mei"] 🡪 isExist returns boolean value

**Looping in Go**

1. basic for:

for i := 0; i < 5; i++ {

fmt.Println("Angka", i)

}

1. for with only conditional argument:

var i = 0

for i < 5 {

fmt.Println("Angka", i)

i++

}

1. for without argument 🡪 use break, continue:

var i = 0

for {

fmt.Println("Angka", i)

i++

if i == 5 {

break

}

}

1. for with range 🡪 like enumerate in python:

var fruits = [4]string{"apple", "grape", "banana", "melon"}

for i, fruit := range fruits {

fmt.Printf("elemen %d : %s\n", i, fruit)

}

for \_, fruit := range fruits {

fmt.Printf("nama buah : %s\n", fruit)

}

for I, \_ := range fruits {

fmt.Printf("elemen %d \n", i)

}

1. for for map 🡪 like for key, item in python:

var satuan = map[string]int{"panjang": 7, "lebar": 4, "tinggi": 6}

for key, item := range satuan {

fmt.Printf("%s = %d\n", key, item)

}

**Function in Go**

1. Basic function, type of inputs and outputs are defined:

func introduction(firstName string, lastName string) (string, string) {

introFirstName := "Hello My First Name Is " + firstName

introLastName := "Hello My Last Name Is " + lastName

return introFirstName, introLastName

}

func main() {

firstName, lastName := introduction("John", "Doe")

}

1. Function with predefined return value:

func tambahAngka(firstNumber int, lastNumber int) (**jumlah** int) {

jumlah = firstNumber+lastNumber

return

}

func main() {

hasil := tambahAngka(4,5)

}

1. Variadic function:

func sum(numbers ...int) int {

var total int = 0

for \_, number := range numbers {

total += number

}

return total

}

func main() {

var total = sum(2, 4, 3, 5, 4, 3, 3, 5, 5, 3) 🡪 regular method

var numbers = []int{2,6,7,8,9,10}

var total = sum(numbers...) 🡪 variadic method

}

1. Combining regular parameter and variadic 🡪 variadic param is located at last:

func yourHobbies(name string, hobbies ...string) {

fmt.Println("Hello, my name is", name)

fmt.Println("My hobbies are: ")

for \_, hobby := range hobbies {

fmt.Println(hobby)

}

}

func main() {

yourHobbies("John", "Gaming", "Hiking", "Reading")

var hobbies = []string{"Sleeping", "Eating"}

yourHobbies("Doe", hobbies...)

}

1. Closure/anonymous function. Closure function assigned to a variable:

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)

}

1. Closure function assigned as return value:

func findMax(numbers []int, max int) (int, func() []int) {

var res []int

for \_, e := range numbers {

if e <= max {

res = append(res, e)

}

}

return len(res), func() []int {

return res

}

}

1. Function assigned as a parameter:

func sayHelloWithFilter(name string, filter func(string) string) {

nameFiltered := filter(name)

fmt.Println("Hello", nameFiltered)

}

func spamFilter(name string) string {

return name // not modifying anything

}

func main() {

sayHelloWithFilter("John", spamFilter)

}

**Pointer in Go**

Pointer is a reference to a memory address. Pointer = memory address

Pointer is like globally changing a variable in the whole code. If not using pointer, the changes made to a variable by a certain function outside the main will not be saved, thus the variable will get no change.

Step:

* In main, declare a variable normally (var var1 string)
* Make a function (func1) that will change the variable, the function’s input’s type is started with ‘\*’ (var1in \*string), everytime the variable is called in the body start it with ‘\*’ (\*var1in)
* In main, when calling the function and inputting the variable, start it with ‘&’ (func1(&var1))

1. Initializing pointer variable:

var number \*int

var name \*string

1. Referencing 🡪 getting the pointer/memory address of a variable 🡪 use &:

var numberA int = 4

fmt.Println("numberA (value) :", numberA) // 4

fmt.Println("numberA (address) :", &numberA) // 0xc20800a220

1. Dereferencing 🡪 getting the value of a pointer/memory address 🡪 use \*:

var numberA int = 4

var numberB \*int = &numberA

fmt.Println("numberB (value) :", \*numberB) // 4 🡪 calling numberB still needs \*

fmt.Println("numberB (address) :", numberB) // 0xc20800a220

1. Change a pointer variable, change other variables with same memory address:

var numberA int = 4

var numberB \*int = &numberA

numberA = 5 // this will change numberA and numberB value

\*numberB = 6 // this will change numberA and numberB value

1. Pointer as parameter:

func change(original \*int, value int) {

\*original = value

}

func main() {

var number = 4

fmt.Println("before :", number) // 4

change(&number, 10)

fmt.Println("after :", number) // 10

}

**Struct in Go**

Struct data type = Class. Struct consists of variables (properties in class) and functions (methods in class)

1. Declaring struct:

type student struct {

name string

grade int

}

1. Instantiating struct:

func main() {

var john student

john.name = "john doe"

john.grade = 2

}

1. Struct literals:

// cara 1

var john = student{}

john.name = "wick"

john.grade = 2

// cara 2

var doe = student{"doe", 2}

// cara 3

var jack = student{name: "jack", grade: 2}

1. Embedded struct 🡪 struct as a type of another struct’s parameter:

type person struct {

name string

age int

}

type student struct {

grade int

person

}

func main() {

// contoh 1

var john = student{}

john.name = "john"

john.age = 21

john.grade = 2

// contoh 2

var doeData = person{name: "doe", age: 21}

var doe = student{person: doeData, grade: 2}

}

1. Anonymous Struct 🡪 declared on line when instantiating object, nice for single use:

func main() {

var john = struct {

name string

grade int

}{} 🡪 {} as initialization is compulsory, or fill it with the needed properties

}

1. Nested struct 🡪 anonymous struct that is embedded to a struct

type student struct {

person struct {

name string

age int

}

grade int

}

1. Method 🡪 function owned by a struct:

type student struct {

name string

grade int

}

func (s student) sayHello() {

fmt.Println("halo", s.name)

}

func main() {

var john = student{"john wick", 21}

john.sayHello()

}

1. Struct’s method’s parameter as pointer, so struct’s property change will be kept:

type phone struct {

name string

year int

}

func (p \*phone) changeYear(newYear int) { 🡪 add \* here

p.year = newYear

}

func main() {

var phoneA = &phone{name: "iPhone 13", year: 2022} 🡪 add & here

phoneA.changeYear(1999)

fmt.Println(\*phoneA) 🡪 add \* here

}

**Interface in Go**

Interface is a data type in go. It consists of empty methods. An interface variable can be assigned with an instance object of a struct data type.

1. Making interface:

type hitung interface {

luas() float64 🡪 the interface has the structs’ method

}

type persegi struct {

sisi float64

}

func (p persegi) luas() float64 {

return math.Pow(p.sisi, 2)

}

type lingkaran struct {

diameter float64

}

func (l lingkaran) jariJari() float64 {

return l.diameter / 2

}

func (l lingkaran) luas() float64 {

return math.Pi \* math.Pow(l.jariJari(), 2)

}

func main() {

var bangunDatar hitung = persegi{10.0} 🡪 interface as type, while also instantiating the struct

fmt.Println("luas :", bangunDatar.luas())

bangunDatar = lingkaran{14.0}

fmt.Println("luas :", bangunDatar.luas())

}

1. Embedded interface 🡪 calling other interfaces to an interface:

type hitung2d interface {

luas() float64

}

type hitung3d interface {

volume() float64

}

type hitung interface {

hitung2d

hitung3d

}

type kubus struct {

sisi float64

}

func (k kubus) luas() float64 {

return math.Pow(k.sisi, 2) \* 6

}

func (k kubus) volume() float64 {

return math.Pow(k.sisi, 3)

}

func main() {

var bangunRuang hitung = kubus{4}

fmt.Println("luas :", bangunRuang.luas())

fmt.Println("volume :", bangunRuang.volume())

}

1. Empty interface 🡪 as data type that can be assigned by any type of value (int, arr, etc), but have to be casted if the value want to be processed:

var secret interface{} 🡪 declared as interface{} data type

secret = 2

var number = secret.(int) \* 10 🡪 casting to int before use

secret = []string{"apple", "manggo", "banana"}

var fruits = strings.Join(secret.([]string), ", ") 🡪 casting to array of string before use

1. Casting empty interface to pointer:

type person struct {

name string

age int

}

func main(){

var secret interface{} = &person{name: "wick", age: 27} 🡪 use & to reference address

var name = secret.(\*person).name 🡪 use \* to dereference (call the value)

}

**Package in Go**

One folder has one package for all its files

Every component (func, struct, var) in one package can be used by all files

Two kinds of components right in Go:

1. Public/Exported 🡪 can be accessed by other package 🡪 component’s name started with Capital case. What to capitalize when importing exported components:

In exported package (library):

* struct’s name, properties
* interface’s name, method
* method’s name, its struct’s name, its struct’s properties’ names called in method
* func’s name only

In importing package (main):

* all variables that are Capital in imported package

1. Private/Unexported 🡪 only accessed by same package 🡪 component’s name started with non Capital case

To import package and use component in another package:

package library

func SayHello(name string) {

fmt.Println("hello")

}

1. Regular method:

package main

import "belajar-golang-level-akses/library"

func main() {

library.SayHello("john")

}

1. Using other package without prefix:

package main

import . "belajar-golang-level-akses/library"

func main() {

SayHello("john")

}

1. Using other package with alias. A struct’s methods does not need prefix in main:

package main

import b "belajar-golang-level-akses/library"

func main() {

b.SayHello("john")

}

1. Using multiple files (file1.go, file2.go) with same package (package main) 🡪 run all files:

go run file1.go file2.go

To automatically run a function when a package being called 🡪 create a func init():

At other package:

package library

var DatabaseConnection string

func init() {

DatabaseConnection = "MySQL"

fmt.Println("Melakukan init")

}

At main package:

package main

import "belajar-golang-level-akses/library"

func main() {

library.DatabaseConnection)

}

If only want to run init func without running other funcs (Go will complaint if func imported not used):

package main

import \_ "belajar-golang-level-akses/library"

func main() {

}

**External Package in Go**

To install external package, create a Go project first, then type in terminal:

go mod init api-mysql

go get github.com/go-sql-driver/mysql

go get github.com/julienschmidt/httprouter

There will be add on in go.mod files:

module api-mysql

go 1.17

require (

github.com/go-sql-driver/mysql v1.6.0 // indirect

github.com/julienschmidt/httprouter v1.3.0 // indirect

)

**Defer function in Go**

1. Defer 🡪 Will always run after a function run eventhought that function error:

func logging(){

fmt.Println("selesai memanggil function")

}

func runApplication(){

defer logging()

fmt.Println("Run Application")

}

func main(){

runApplication()

}

1. Panic 🡪 Will run after an error happen in a function, program will stop. Defer function will still run:

func endApp(){

fmt.Println("End App")

}

func runApp(error bool){

defer endApp()

if error{

panic("ERROR")

}

}

func main(){

runApp(true)

}

1. Recover 🡪 panic proses will stop, program will continue to run, recover get the panic’s message:

func endApp(){

fmt.Println("End App")

message := recover()

fmt.Println("Terjadi Error", message)

}

func runApp(error bool){

defer endApp()

if error{

panic("ERROR")

}

}

func main(){

runApp(true)

}

1. Error 🡪 type in Go, many internal functions return error along with the output:

func main() {

var input1 string = p

number, err := strconv.Atoi(input)

fmt.Println(err.Error()) // strconv.Atoi: parsing "p": invalid syntax

}

Custom error message:

func pembagian(nilai uint, pembagi uint)(float64, error){

if pembagi == 0 {

return 0.0, errors.New("Maaf pembagi tidak boleh NOL")

}else{

return float64(nilai/pembagi), nil

}

}

func main(){

hasil, err := pembagian(8,4)

if err == nil{

fmt.Println("Hasil", hasil)

}else{

fmt.Println("Error", err.Error())

}

}

**Golang Packages**

1. os 🡪 to access operating system’s fitures:

import "os"

func main() {

args := os.Args

fmt.Println("Argument : ", args) // Argument : [C:\Users\ASUS\AppData\Local\Temp\go-build2306494301\b001\exe\tugas10.exe]

name, err := os.Hostname() // Hostname : LAPTOP-981ISP11

fmt.Println("Hostname : ", name)

}

1. flag 🡪 implements command-line flag parsing. Like arguments, parameterize program execution, written in key-value pairs:

import "flag"

func main() {

var name = flag.String("name", "anonymous", "type your name")

var age = flag.Int64("age", 25, "type your age")

flag.Parse()

fmt.Printf("name\t: %s\n", \*name)

fmt.Printf("age\t: %d\n", \*age)

}

1. math 🡪 math opeations:

math.Round(float64) 🡪 rounding float64 up or down, based on nearest integer

math.Floor(float64) 🡪 rounding float64 down

math.Ceil(float64) 🡪 rounding float64 up

math.Max(float64, float64) 🡪 returning max float64 value

math.Min(float64, float64) 🡪 returning min float64 value

1. time 🡪 time operations:

time.Now() 🡪 returning current time

time.Date(...) 🡪 to create date

time.Parse(layout, string) 🡪 Parse date from string

time.Sleep() 🡪 To stop (Sleep) program for a while, combine with time.Second

import "time"

func main() {

now := time.Now()

fmt.Println(now)

utc := time.Date(2020, 10, 10, 10, 10, 10, 10, time.UTC)

fmt.Println(utc)

layout := "2006-01-02"

parse, \_ := time.Parse(layout, "1990-03-20")

fmt.Println(parse)

fmt.Println("start")

time.Sleep(time.Second \* 3)

fmt.Println("after 3 seconds")

}

1. sort 🡪 for sorting

import "sort"

func main() {

s := []string{"Go", "Bravo", "Gopher", "Alpha", "Grin", "Delta"}

sort.Strings(s)

fmt.Println(s) // [Alpha Bravo Delta Go Gopher Grin]

}

1. reflect 🡪 to see the internal structure of the code when the code is running

import "reflect"

func main() {

var number = 23

var reflectValue = reflect.ValueOf(number)

fmt.Println("tipe variabel :", reflectValue.Type()) // tipe variabel : int

if reflectValue.Kind() == reflect.Int {

fmt.Println("nilai variabel :", reflectValue.Int()) // nilai variabel : 23

}

}

**Goroutine**

Goroutine 🡪 lightweight thread, asynchronous (will not wait for other goroutine)

To make a goroutine, start a func or method with ‘go’:

func say(s string) {

for i := 0; i < 5; i++ {

time.Sleep(100 \* time.Millisecond)

fmt.Println(s)

}

}

func main() {

go say("world")

say("hello")

}

**Channel**

Channel 🡪 connecting between goroutines, send and receive mechanism

1. To declare a channel, the data type is ‘chan’, in func and in main (the name can be different), also declare the send and receive:

angka := make(chan int)

a <- 10 // to send, assigning/giving channel a with 10, located at func

angka := <- a // to receive, located at main

Example code:

func cetak(ch chan int) {

fmt.Println("ini dari goroutine...")

ch <- 10

}

func main() {

angka:= make(chan int)

go cetak(angka)

nilai := <-angka

fmt.Println("nilai channel integer :", nilai)

fmt.Println("ini dari function main...")

}

1. Make an infinite loop to get data from goroutine, then close it when goroutine stop:

func cetak(ch chan<- int) {

for index := 0; index < 10; index++ {

ch <- index

}

close(ch)

}

func main() {

angka := make(chan int)

go cetak(angka)

for { // this for is infinite, receive output of cetak() that runs on goroutine thread

data, ok := <-angka

if ok == false {

break

}

fmt.Printf("Data di terima %v\n", data)

}

}

1. Make an infinite loop to get data from goroutine, then close it when goroutine stop:

func cetak(ch chan<- int) {

for index := 0; index < 10; index++ {

ch <- index

}

close(ch)

}

func main() {

angka := make(chan int)

go cetak(angka)

for { // this for is infinite, receive output of cetak() that runs on goroutine thread

data, ok := <-angka

if ok == false {

break

}

fmt.Printf("Data di terima %v\n", data)

}

}

1. Make a channel buffer so that data transfer not done 1 by one, but in a buffer of n size:

func main() {

ch := make(chan int, 3)

ch <- 6

ch <- 7

ch <- 5

fmt.Println(<-ch)

fmt.Println(<-ch)

fmt.Println(<-ch)

}

1. Using multiple channels with selec:

func getAverage(numbers []int, ch chan float64) {

var sum = 0

for \_, e := range numbers {

sum += e

}

ch <- float64(sum) / float64(len(numbers))

}

func getMax(numbers []int, ch chan int) {

var max = numbers[0]

for \_, e := range numbers {

if max < e {

max = e

}

}

ch <- max

}

func main() {

var numbers = []int{3, 4, 3, 5, 6, 3, 2, 2, 6, 3, 4, 6, 3}

fmt.Println("numbers :", numbers)

var ch1 = make(chan float64)

go getAverage(numbers, ch1)

var ch2 = make(chan int)

go getMax(numbers, ch2)

for i := 0; i < 2; i++ {

select {

case avg := <-ch1:

fmt.Printf("Avg \t: %.2f \n", avg)

case max := <-ch2:

fmt.Printf("Max \t: %d \n", max)

}

}

}

1. WaitGroup 🡪 synchronize multiple goroutine:

* Adding() 🡪 to add running goroutine
* Wait() 🡪 to wait the running process
* Done() 🡪 to sign that all goroutines have finished

Goroutine process is so fast that making process inside the func cannot be seen in the output, e.g. when making print in the func. If the process only outputting to channel, we don’t have to print it. But if the process needs to print, the output cannot be seen. To solve it, the conventional way is by using sleep. We can not having to use sleep by using WaitGroup:

import "sync"

func printText(text string, wg \*sync.WaitGroup){

for i:=0; i<5; i++{

fmt.Println(text)

}

wg.Done()

}

func main(){

var wg sync.WaitGroup

wg.Add(1)

go printText("Halo", &wg)

wg.Add(1)

go printText("Dunia", &wg)

wg.Wait()

}

**JSON in Go**

In Go, JSON is written as string

Decode json to readable object/data type using json.Unmarshal:

1. Method 1, decode json to object struct:

import "encoding/json"

type User struct {

FullName string `json:"Name"` // casting ‘Name’ to ‘FullName’

Age int // all properties must be public (capitalized)

}

func main() {

var jsonString = `{"Name": "john doe", "Age": 27}` // because here typed ‘Name’

var jsonData = []byte(jsonString) // json.Unmarshal only take byte data type, so do this

var data User // this var will receive json.Unmarshal output

json.Unmarshal(jsonData, &data) //json.Unmarshal returns the output to ponter var

fmt.Println("user :", data.FullName) // user : john doe

fmt.Println("age :", data.Age) // age : 27

}

1. Method 2, decode json to map[string]interface{}:

var data1 map[string]interface{}

json.Unmarshal(jsonData, &data1)

fmt.Println("user :", data1["Name"]) // interface will take the map data type of json data1

fmt.Println("age :", data1["Age"])

1. Method 3, decode json to interface{}, then cast it to map[string]interface{}:

var data2 interface{} // interface take any data type including raw json

json.Unmarshal(jsonData, &data2)

var decodedData = data2.(map[string]interface{}) // but must be casted into map data type

fmt.Println("user :", decodedData["Name"])

fmt.Println("age :", decodedData["Age"])

1. Decode array json to array object 🡪 similar to above methods but in array

import "encoding/json"

type User struct {

FullName string `json:"Name"`

Age int

}

func main() {

var jsonString = `[

{"Name": "john doe", "Age": 27},

{"Name": "doe john", "Age": 32} ]`

var data []User

json.Unmarshal([]byte(jsonString), &data)

fmt.Println("user 1:", data[0].FullName)

fmt.Println("user 2:", data[1].FullName)

}

Encode object to json using json.Marshal 🡪 input can be of types object struct, map[string]interface{}, or slice:

1. Encode object struct to json (string):

import "encoding/json"

type User struct {

FullName string `json:"Name"`

Age int

}

func main() {

var object = User{"john doe", 27}

var jsonData, err = json.Marshal(object) // output json.Marshal is of type []byte

var jsonString = string(jsonData) // so convert to string first

fmt.Println(jsonString) // {"Name":"john doe","Age":27}

}

1. Encode array of object struct to json (string):

var object = []User{{"john doe", 27}, {"doe john", 32}}

var jsonData, err = json.Marshal(object) // output json.Marshal is of type []byte

var jsonString = string(jsonData) // so convert to string first

fmt.Println(jsonString) // [{"Name":"john doe","Age":27},{"Name":"doe john","Age":32}]

**Web Server in Go**

Can be made by using internal package net/http, not like other languages that needs install libraries

Package net/http features 🡪 web server, routing, templating (for view), start server

To start a simple web server:

import "net/http"

func index(w http.ResponseWriter, r \*http.Request) {

fmt.Fprintln(w, "apa kabar!")

}

func main() {

http.HandleFunc("/", func(w http.ResponseWriter, r \*http.Request) {

fmt.Fprintln(w, "halo!") // will be show at http://localhost:8080/

})

http.HandleFunc("/index", index) // will be show at http://localhost:8080/index

http.ListenAndServe(":8080", nil) // to live the server

}

http.HandleFunc(route, callback function) 🡪 for routing management

**Web service API server in Go**

Web service API 🡪 web that receive request from client and return response, usually in json/xml

1. GET request 🡪 postman/client request data to golang server, data shown in postman:

type Movie struct {

ID int `json:"id"`

Title string `json:"title"`

Year int `json:"year"`

}

func Movies() []Movie {

movs := []Movie{

{1, "Spider-Man", 2002},

{2, "John Wick", 2014},

{3, "Avengers", 2018},

{4, "Logan", 2017},

}

return movs

}

// GetMovies

func getMovies(w http.ResponseWriter, r \*http.Request) { // w 🡪 write: header, body, etc

if r.Method == "GET" { // r 🡪 request: header, body, etc

movies := Movies()

dataMovies, err := json.Marshal(movies)

if err != nil {

http.Error(w, err.Error(), http.StatusInternalServerError)

}

w.Header().Set("Content-Type", "application/json")

w.WriteHeader(http.StatusOK)

w.Write(dataMovies)

return

}

http.Error(w, "ERROR....", http.StatusNotFound)

}

func main() {

http.HandleFunc("/movies", getMovies)

}

1. POST request 🡪 postman/client post data to golang server, data shown back in postman:

// PostMovie

func PostMovie(w http.ResponseWriter, r \*http.Request) {

w.Header().Set("Content-Type", "application/json")

var Mov Movie

if r.Method == "POST" {

if r.Header.Get("Content-Type") == "application/json" { // header’s content type

// parse from json

decodeJSON := json.NewDecoder(r.Body)

if err := decodeJSON.Decode(&Mov); err != nil {

log.Fatal(err)

}

} else { // header’s content type = form-data

// parse dari form

getID := r.PostFormValue("id")

id, \_ := strconv.Atoi(getID)

title := r.PostFormValue("title")

getYear := r.PostFormValue("year")

year, \_ := strconv.Atoi(getYear)

Mov = Movie{

ID: id,

Title: title,

Year: year,

}

}

dataMovie, \_ := json.Marshal(Mov) // to byte

w.Write(dataMovie) // cetak di browser

return

}

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

return

}

func main() {

http.HandleFunc("/post\_movie", PostMovie)

}

**Middleware in Go**

Middleware 🡪 a process that must be done between routing’s request and its callback function, to decide whether stop or continue the request 🡪 e.g. in authenticating token

1. Middleware for simply checking url parameter “token”:

func CekLogin(next http.Handler) http.Handler {

return http.HandlerFunc(func(w http.ResponseWriter, r \*http.Request) {

if r.URL.Query().Get("token") != "12345" {

fmt.Fprintf(w, "Token tidak tersedia atau salah\n")

return

}

next.ServeHTTP(w, r)

})

}

func GetMovie(w http.ResponseWriter, r \*http.Request) {

w.Write([]byte("<h1>Anda Berhasil Mengakses Fungsi GetMovie() </h1>"))

}

func main() {

http.Handle("/", CekLogin(http.HandlerFunc(GetMovie)))

}

1. Middleware for basic auth 🡪 checking user, password, and status ok:

func Auth(next http.Handler) http.Handler {

return http.HandlerFunc(func(w http.ResponseWriter, r \*http.Request) {

uname, pwd, ok := r.BasicAuth()

if !ok {

w.Write([]byte("Username atau Password tidak boleh kosong"))

return

}

if uname == "admin" && pwd == "admin" {

next.ServeHTTP(w, r)

return

}

w.Write([]byte("Username atau Password tidak sesuai"))

return

})

}

func GetMovie(w http.ResponseWriter, r \*http.Request) {

if r.Method == "POST" {

w.Write([]byte("<h1>Anda Berhasil Mengakses Fungsi GetMovie() </h1>"))

}

}

func main() {

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

}