**GOLANG**

**Basic**

**What is Go?**

GO is an open source programming language which makes it easy to build simple, reliable and efficient software. Programs are constructed from packages, whose properties allow efficient management of dependencies.

**Why should you learn Go?**

* go is easy to learn, easy of use
* go has an easy to use interface
* great for web development, even front end

Go is good for performance because it is a compiled language, giving it a performance which is on-par with native C code. Go provides many of the high-level features such as garbage-collection. Go also integrates concurrent programming concepts which a great advantage when programming for multi-threaded, multi-core platforms.

**What is Go suitable for?**

**The really good: Network and Web server:**

* Network applications live and die by concurrency, and Go concurrency have [goroutines](http://www.infoworld.com/article/2683845/google-go/164121-Fast-guide-to-Go-programming.html#slide7) and [channels](http://www.infoworld.com/article/2683845/google-go/164121-Fast-guide-to-Go-programming.html#slide8), are well suited for such work

**Stand-alone command-line apps or scripts:**

* Another advantage Go has here is speed. The resulting executables run far faster than Python

**Explain advantage and disadvantage of Go?**

**Advantage:**

T think Golang has more advantage:

* **Golang is open source:** a programming language to be open-source to become better, cleaner, and more efficient
* GO compiles very quickly becauseGo has a simple structure and syntax so it provides faster feedback, shorter time and saves time for development.
* **Golang is concurrent:** Golang has efficient concurrency, like C, C++, Java, and at the same time concurrency in Go is done much easier and faster with goroutines, channels
* **Golang has a garbage collector:** good for performance and helps to make concurrency more efficient.

**Disadvantages :**

* No generics support
* Err everywhere. You need to check errors for each of the error producing function in your code explicitly

**How Go is different from the programming languages that you have used? (Java): different**

* Java is slow - Golang is known to be one of the fastest languages because Go is simpler structure and syntax
* The interface in Go is more flexible with Java**.**
* Java use VM(Virtual machine), Go use native code so Go’s compilation speed is very fast
* Go support go-routine in built-in library which very good for developer with me
* Java doesn't support return of more than one value and Go has this
* Java takes a lot of memory space, Golang doesn’t

## Array and Slice

## Array:

## Len and cap always is equal

## Size is fixed and can’t add new item

## Slice:

## Slice is much more flexible, powerful, and convenient than arrays. Unlike arrays, slices can be resized

## Size can be extended

## Array := [2000]int{1,2,3}

## S1 := append([]int{}, array[:2]…) that gabage collection

## Can’t compare 2 slice

## Byte.Equal()

## Sort.Ints(S)

## Use interface sort

## Map:

## Given a key and a value, you can strore the value in a Map object. After value is stored, you can retrieve it by using its key

**What are the differences between slice and link list?**

* The major difference between Array and Linked list regards to their structure. Each element of Array is associated with an index. About Linked list each node of LK the references to the previous and next element
* Array has a fixed size and required to be declared prior, but Linked List is not fixed size and expand and link together during execution.

**How re- slicing process of a slice happen under the hood?**

**Can you modify array length? Can you append new item into an array?**

* Can’t modify array length

## String

**How is string in Go constructed? And what is the different between string in Go with other programming languages**

* Strings are defined between double quotes "..." and not single quote
* If you use [**range**](https://gobyexample.com/range) within for loop, range will return rune and **byte index** of the character-
* Use ` ` (raw string literals and interpreted string literals(“”)) that down the line often use to in regex and html
* Check is string : str.Validstring()
* Str.Trim()
* Covert string -> Int: strconv.Atoi(str)

**What is string literal? How is it done in Go?**

## Goroutine

**What are the difference between goroutine and OS thread?(N)**

* Goroutines is easy to use than OS thread
* OS is managed by the operating system
* Goroutine is managed by goroutime so can run in one theard
* Synchronization, which the OS must do, is expensive than goroutine
* Goroutine no fix size when we start so very light

**Explain what is go routine in GO? How you can stop go routine**?

* A goroutine is a function or method that is running in background concurrently with other goroutine
* GO provides a special keyword “go” to create goroutine. When we call a function or method with “go” prefix, goroutine will be excuted
* Goroutine is key feature to bring you to concurrency in Go
* A goroutine is a lightweight thread of execution: multiple goroutines can be run inside 1 OS thread.
* To stop goroutine, you pass the goroutine  a signal channel, that signal channel is used to push a value into when you want the goroutine to stop.  The goroutine polls that channel regularly. As soon as it detects a signal, it quits
* GOMAXPROCS

**What is a worker pool? And how to do that in Go?**

Many woker work on many work

A worker pool in my opinion is a grouping of workers (threads, processes, coroutines) many woker work on many work, that are running the same general task definition, or code reading from a queue of work tasks.

* What I like to think of is a fast food restaurant when thinking about worker pools. Imagine your code is a fast food restaurant. You have a customer walk up to the counter and order a hamburger
* his worker takes the order, runs back to the kitchen, makes the burger, brings the burger back to the customer, and repeats for the next customer. This is such a terrible restaurant that you leave while waiting for hours in line to get your order in, and then you blog about it and the restaurant goes out of business. Failure**.**
* “n” workers take “n” customer orders all at once, so each customer order is handled as they come in, and everyone is happy

**How to stop a worker pool?**

**What is fan-in, fan-out?**

* It is a design pattern
* When you need to merged multi chanel to on one channel, you use fan-in
* When you want one input is procced by multi goroutine, you use fan-out

**What is sync.WaitGroup used for?**

- sync.WaitGroup provides a goroutine synchronization mechanism in Golang, and is used for waiting for a collection of goroutines to finish.

- wg.Add()- wg.Done()- wg.Wait()

**What are the differences between concurrency and parallelism?**

* Currency is when two or more tasks can start, run and complete in overlapping time
* Parallelism is when tasks run at the same time

**What are the differences between sync.Mutex and semaphore?**

* There are two methods defined on [Mutex](https://tip.golang.org/pkg/sync/" \l "Mutex) namely [Lock](https://tip.golang.org/pkg/sync/#Mutex.Lock) and [Unlock](https://tip.golang.org/pkg/sync/#Mutex.Unlock)
* Semaphore is an integer variable. Mutex is an object
* A Mutex is used to provide a locking mechanism to ensure that only one Goroutine is running the critical section of code at any point of time to prevent race condition can happen.
* We can understand that only one process or goroutine at a time can access the given resource to avoid conflicts. Semaphore allows multiple program threads to access the resource
* When we don't need communication among goroutines we use **sync.Mutex**
* Semaphore value is modified Mutex object is locked or unlocked by the process requesting or releasing the resource.
* Semaphore is a signalling mechanism. Mutex is a locking mechanism
* A semaphore is a generalized mutex.we can split the 4 KB buffer into four buffers with each 1kb for 1 buffer (identical resources). A semaphore can be associated with these four buffers. The consumer and producer can work on different buffers at the same time

**Race condition:**

* A **race condition** is when two or more routines have access to the same resourceor variable. Using sync.Mutex or channel

### **Blocking with waitgroups**

* Go run –race main.go

**What are the differences between sharing by communicate vs communicate by sharing?**

* "Don't communicate by sharing memory; share memory by communicating" ~ "Đừng trao đổi bằng cách chia sẻ vùng nhớ; chia sẻ vùng nhớ bằng trao đổi".
* Mutex: mutex.lock() and mutex.unclock()
* Go run – race main.go: check goroutine to fix the same value in 1 time

**What is channel? How does it work? What is it used for?**

* Channels are the pipes that connect concurrent goroutines
* Channels make goroutines share memory by communicating
* One goroutine can send data into a channel, while other goroutines can receive that data from the same channel.
* firt in, firt out
* Remember close channel
* Multiplexing channel with select: recive the result from mongo, elasticsearch or goolgle, with the result is quick, it is accepted first
* Time.After()

### Syntax

### Create a new channel: channel := make(chan type\_of\_channel)

* Put value into a channel: channel <- value
* Take value from a channel: value := <-channel
* Close a channel: close(channel)
* Range over a channel:  for value := range channel

**What is context? What is it used for?**

**- c**ontext.TODO(): is recommended when we don't know what to use for any something

- context.WithTimeout(**c**ontext.TODO(), time)

- context.WithDeadline()

- context.CancelFunc(0

- cancel()-

- Context packages are often used for service APIs.

* A way to think about context package in go is that it allows you to pass in a “context” to your program. This is where a timeout or deadline context can handler. Context like a timeout or deadline or a channel to indicate stop working and return. if you are doing a web request or running a system command, it is usually a good idea to have a timeout for your systems

**Have you used goroutine in your application? Describe it.**

**Why goroutine is better than thread?**

* You can run more goroutines on a typical system than you can with threads.
* Goroutine no fix size when you start so Goroutines have a faster startup time than threads.
* Goroutines come with built-in primitives to communicate safely between themselves (channels).

**What is goroutine leak in Go? How to prevent it?**

* A common type of memory leak is leaking Goroutines. When you start a Goroutine that you expect to will stop it but it never happens then it has leaked.
* It lives for the lifetime and memory of your application. So I think “Never start a goroutine if you don’t know how it will stop”?
* Prevent it you can use: package context or With done

## Interface

**Explain Interface in Go**

* we need to use type alias to simplify interface declaration along with keyword interface
* To implement an interface in Go, we just need to implement all the methods in the interface
* Interfaces make the code more flexible
* In GO, interfaces is a way to specify the behaviour of an object

**Struct**

* Struct is a user-defined type that contain a collection of name fiel/properties. It used to group related data together to from a single unit

**How interface in Go different from interface in other languages?**

* Go does not have generic types, protected inheritance, static members, abstract types, final members so those annotations in Java interfaces are unavailable
* Java interfaces can provide a default implementation for methods that an implementing class may choose not to override. Go interfaces cannot.

**When should you use interface?**

**How did you use interface in your previous project?**

* Behavior for object

## HTTP

### **Router:**

* Router is a component that routes your request to target handler base on a predefined rules

### **Handler:**

* Handler is where we implement our business logic.
* A HTTP handler in Go is an interface witch has exactly one method ServeHTTP which will process the http.Request and response to client via the http.ResponseWriter.
* So we can create our custom handler easily by create a type that implement the http.Handler interface.

**HTTP:** Go provides a rich features support of HTTP. It has everything you need to work with HTTP from a server, a router, handler, encoding or even a HTML template where you can use to do some server-side rendering website.

To have a complete HTTP application as what we listed in previous section, we will need:

* Create a handler that handle our logic
* Create a router and register the handler
* Create an HTTP server and register the router

1. **Have you ever used any framework for working with HTTP in Go?**

**Gorrila Mux**

1. **What is middleware? Have you ever written one?**

Middleware is simply a handler stay in between the client and the target handler which will normally do some works on the request and the response of a HTTP request.

* Middleware can be applied in global scope or can apply to a specific route. It can be used to process request or response of a HTTP request. Below are some examples of middleware:
* Logging request info
* Logging HTTP response, status, time consumption
* Authentication validation
* [Cross-Origin Resource Sharing](https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS)
* Caching
* Size limiting
* Rate limiting
* Transform the response to be standard response before respond to client.
* Router -> middleware -> handler (http.request)
* Handler -> midlderware -> router -> client(http.ReponseWriter)

## Performance

1. How many query/second of your application?
2. How can you measure performance of your application?
3. How do you scale up your application?

## Others

**Does Go pass by value or pass by references? What are the differences?**

**What is Go run time?**

* the Go runtime implements all the built into the Go language: goroutines, memory allocation, the garbage collector, channels, interfaces, maps, slices.
* Package runtime contains operations that interact with Go's runtime system, such as functions to control goroutines

**How does garbage collector work in Go?**

* Garbage Collection is the process of freeing memory space that is not being used. In other words, the garbage collector sees which objects are out of scope and cannot be referenced any more and frees the memory space they consume. This process happens in a concurrent way while a Go program is running, not before or after the execution of a Go program. The operation of the Go GC is based on the tricolor algorithm.

**What is inside the interface{}?**

**How to handle gracefully shutdown in Go?**

**Why would you prefer to use an empty struct{} ? Provide some examples of the good use of the empty struct{}**

**How do you compare two structs? What about two interfaces? Provide examples.**

**How do you manage configurations of your services in Go?**

**How do you structure your Go projects? Explain the advantages and disadvantages?**

* Cycle list
* I use the structure of Standard Go Project Layout one of on structure popular on Github
* README: required install software for backend and front end, detail project that you know project use to for what technical and use to for what target. Easy transfer for new member
* Version: use for Jenkin, Build version
* Main.go: use to execute application
* Make file: **makefile** is the build step, include all command need to development often use for project ex build docker or test. That sure command can run evn Dev and evn Production
* File LICENSE
* Folder Deployment:  system and container deployment configurations and templates (docker-compose, kubernetes/helm, mesos, terraform, bosh).
* Folder Script: This folder is optional, Scripts to perform various build, install, analysis, etc operations. These scripts keep the root level Makefile small and simple
* Folder internal: Private application and library code. This is the code you don't want others importing in their applications or libraries.
* Folder /pkg: Library code that's ok to use by external applications. Other projects will import these libraries expecting them to work. This is place contain my code (/internal)

+ Types: contain all model can share for an other package

### Folder /configs:C onfiguration file templates or default configs.

### /web:Web application specific components: static web assets, server side templates and SPAs.

* Hander 🡨 service 🡨 repository
* Hander 🡨 external 🡨 internal 🡨 mongodb

**Iota:**

* When not have value it will get the last near value

**Panic:**

* **Panic** is a built-in function that stops the ordinary flow of control and begins *panicking*. When the function F calls panic, execution of F stops, any deferred functions in F are executed normally, and then F returns to its caller. To the caller, F then behaves like a call to panic. The process continues up the stack until all functions in the current goroutine have returned, at which point the program crashes. Panics can be initiated by invoking panic directly. They can also be caused by runtime errors, such as out-of-bounds array accesses

**Defer:**

* A **defer statement** pushes a function call onto a list. The list of saved calls is executed in *Last In First Out order*after the surrounding function returns. Defer is commonly used to simplify functions that perform various clean-up actions

**Recover:**

* **Recover** is a built-in function that regains control of a panicking goroutine. Recover is only useful inside deferred functions. During normal execution, a call to recover will return nil and have no other effect. If the current goroutine is panicking, a call to recover will capture the value given to panic and resume normal execution.

**GoTest**

go test -v -run=" string name funtion " : test theo ten function

create a file with a name that ends in \_test.go

* TestXxx: not starting with lowercase letter
* It takes a single parameter that is an instance of \*testing.T

**The step build docker file:**

* Set the original image: I need to declare the parent of Image. I use key work “From”. I can use Docker-hub that download image necessary. MAINTAINER: An optiona l used to declare the author of Dockerfile
* We will install additional applications, set up the necessary environment:
* Run: To execute a command during build images.
* CMD: To execute a command when container start. If have more CMD, It will execute the last CMD
* ENTRYPOINT: Để thực thi một số câu lệnh trong quá trình start container, những câu lệnh này sẽ được viết trong file .sh.
* Configuration:
* Expose: Docker image will listen on the specified port when running
* Add: Copy files, folders, remote files to add them to the filesystem of the image.
* Coppy: Copy files and folders from host machine to image. Can use the url for the file to copy.
* WORKDIR : Định nghĩa directory cho CMD
* Volume: used to share data for containers.