



BASIC GOLANG UNTUK BACKEND DEVELOPER



RAY ANTHONIUS
Backend/Software Engineer
at Verihubs

```
<package main>  
<func Main>  
  <script type="Basic Go-Lang">  
  <script type="Backend Developer">
```



KELAS-WORK
by KELAS.COM

Basic Go-Lang untuk Backend Developer

Perkenalan Bahasa Pemrograman Golang

Go Language atau Go-Lang adalah suatu bahasa pemrograman yang dikelola oleh Google. Go-Lang populer di kalangan “tech” karena *learning curve*-nya yang rendah, namun performanya dapat bersaing dengan pemrograman lain.

Apa saja yang dibutuhkan?

Untuk menggunakan Go-Lang dalam backend dapat menggunakan beberapa aplikasi berikut ini:

- Golang (<https://go.dev/dl/>)
- Code editor (VSCode, Sublime, etc)
- Terminal (command prompt, WSL, etc)
- Git (<https://git-scm.com/downloads>)
- PostgreSQL (<https://www.postgresql.org/download/>)
- Docker (optional)

Mengapa Go-Lang?

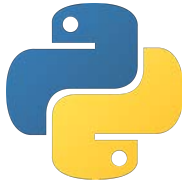
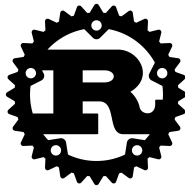

Dari semua bahasa pemrograman yang ada, mungkin banyak yang bertanya, mengapa memilih golang untuk membuat *service*? Mengapa tidak menggunakan rust atau python? Dari sisi kemudahan, python merupakan salah satu bahasa yang paling mudah untuk dipelajari. Sementara di satu sisi, rust memiliki *benchmark* performa yang sangat tinggi dibandingkan bahasa pemrograman yang lain.



KELAS·WORK
by KELAS.COM

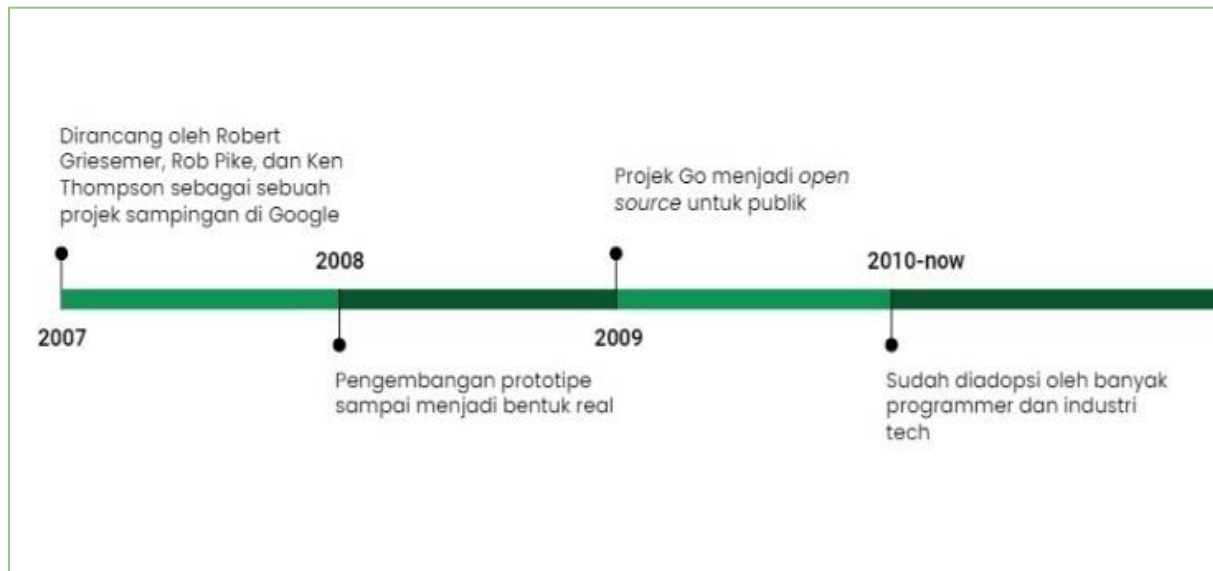
Nyatanya, kedua bahasa tersebut memiliki fokus yang berbeda, dan golang memberikan kemudahan untuk membuat sebuah *service* tanpa mengorbankan performa.

Komparasi bahasa pemrograman

		
Python: <ul style="list-style-type: none">• Mudah dipelajari• Performa kurang• Cocok untuk projek yang cepat	Rust: <ul style="list-style-type: none">• Sulit dipelajari• Performa sangat baik• Bagus untuk sistem besar yang kompleks	Go-Lang: <ul style="list-style-type: none">• Mudah dipelajari• Performa sangat baik• Cocok untuk projek <i>microservices</i>



Sejarah Go-Lang



Go-Lang pertama kali dirancang oleh Robert Griesemer, Rob Pike dan Ken Thompson sebagai sebuah projek sampingan di Google. Lalu pada tahun 2008, ketiga orang tersebut mengembangkan prototipe sampai menjadi bentuk *real*. Dan pada tahun 2009 projek Go dirilis ke publik sebagai bahasa pemrograman *open source*. Semenjak saat itu, Go-Lang telah digunakan oleh programmer dan IT.

Karakteristik Go-Lang

Go-Lang memiliki beberapa karakteristik yang membedakannya dengan bahasa pemrograman lain. Berikut ini merupakan karakteristik dari Go-Lang:

- *Statically Typed*

Variabel yang digunakan harus memiliki tipe data tersendiri. Hal ini bagus untuk menghindari permasalahan saat *compile* (menyusun) program.



KELAS-WORK
by KELAS.COM

- *Compiled*

Semua *code* harus sudah tersusun rapi sebelum bisa dijalankan.

- *Easy to learn & use*

Memiliki struktur yang mudah untuk dipelajari dan nyaman untuk digunakan.

- *High-performance*

Go-Lang dapat menyajikan performa yang sangat baik, jika dibandingkan dengan bahasa pemrograman lain.



KELAS-WORK
by KELAS.COM



KELAS-WORK
by KELAS.COM



Preparation

Set up Go-Lang

Sebelum dapat menggunakan Go-Lang sebagai bahasa pemrograman yang akan dipakai dalam proyek kali ini, terlebih dahulu menginstall dan melakukan set-up Go-Lang di komputer. Berikut cara set-up Go-Lang:

- *Download* Go-Lang dari laman <https://go.dev/dl/>
- Cek instalasi dan verifikasi di go version
- Untuk membuat kode pertama, buatlah main.go
- Dua cara untuk melihat output hasil kode yaitu *go build* dan *go run*
- **Link Go Gitlab untuk project per step:**
<https://github.com/Rocksus/go-restaurant-app>

Primitive Type at Go-Lang

Primitive type merupakan kumpulan tipe-tipe data di Go-Lang yang paling dasar (tidak bisa diturunkan) dan digunakan sebagai bahan bangunan tipe data yang lain.

Tipe data terbagi menjadi empat yaitu:

- 1) Boolean: Berfungsi untuk menyimpan nilai dan benar
- 2) Integer (Int): Digunakan untuk data dengan ukuran minimal 32 bits
- 3) Float: Dapat menyimpan data-data desimal dan hanya bisa dipanggil dengan float 32 dan 64
- 4) String: Hanya bisa menggunakan tanda petik dua (“..”)



Data Type di Go-Lang

- Buatlah example *main.go*
- Dua cara untuk mendeklarasikan variabel yaitu *long declaration* dan *short declaration*
- *Long declaration* menggunakan "var" sebelum tulisan variabel dan memungkinkan untuk dapat tipe data yang kita inginkan
- *Short declaration* adalah *shortcut* dari cara pertama

Turunan Tipe

No	Type	Keterangan
1	Bytes	Alias dari unsigned integer 8 (unit8)
2	Rune	Alias dari int32
3	Complex	Digunakan untuk tipe data yang memerlukan angka nilai imajiner
4	Error	<i>Interface type</i> dari string yang digunakan untuk menangani error di Go-Lang
5	Interface	Digunakan sebagai penentu metode dan penyimpan data kosong



Functions at Go-Lang

- Buatlah *example function main.go*
- Aturan dalam *function* di Go-Lang:
 - 1) Tidak bisa dimulai dengan angka
 - 2) Tidak dapat memiliki spasi
 - 3) Tidak bisa dimulai dengan kata spesial kecuali dengan tanda (`_`)

Pointers

Pointers merupakan sebuah variabel yang menyimpan memori *address*. Sebuah *pointers* bukan merupakan sebuah *value*. *Pointers* dapat digunakan untuk memanggil *address* lain dan mengubah variabel dari jauh menggunakan tanda (*). *Pointers* juga dapat digunakan untuk membuat *MT value* yang bisa keluar 0 atau nil. Simbol yang digunakan untuk mengirim *address* dari variabel adalah tanda (&). Selain itu, *pointers* dapat ditambah dengan data baru dengan kata "new"

Conditionals

- Buatlah *example function main.go*
- Beberapa *conditions* dalam Go-Lang:
 - 1) *if, else if, else* digunakan untuk kondisi yang memiliki syarat
 - 2) *Switch* digunakan untuk menghitung angka dengan value ABCD
 - 3) *Looping*. Ada tiga komponen loop yaitu: `i=0, i<5, i++`



Methods & Interface

- Buatlah *example function main.go*
- *Interface* bisa berfungsi sebagai penyimpan data kosong atau
- *Interface* juga berguna sebagai penentu fungsi-fungsi yang harus diimplementasi
- Implementasi di Go-Lang menggunakan *method*
- *Method* bisa digunakan oleh dua hal yaitu *type alias* dan *struct*
- *Type alias* adalah tipe data baru yang kita ambil dari tipe lain
- *Struct* adalah kumpulan beberapa variabel menjadi sebuah objek yang bisa kita gunakan berkali-kali

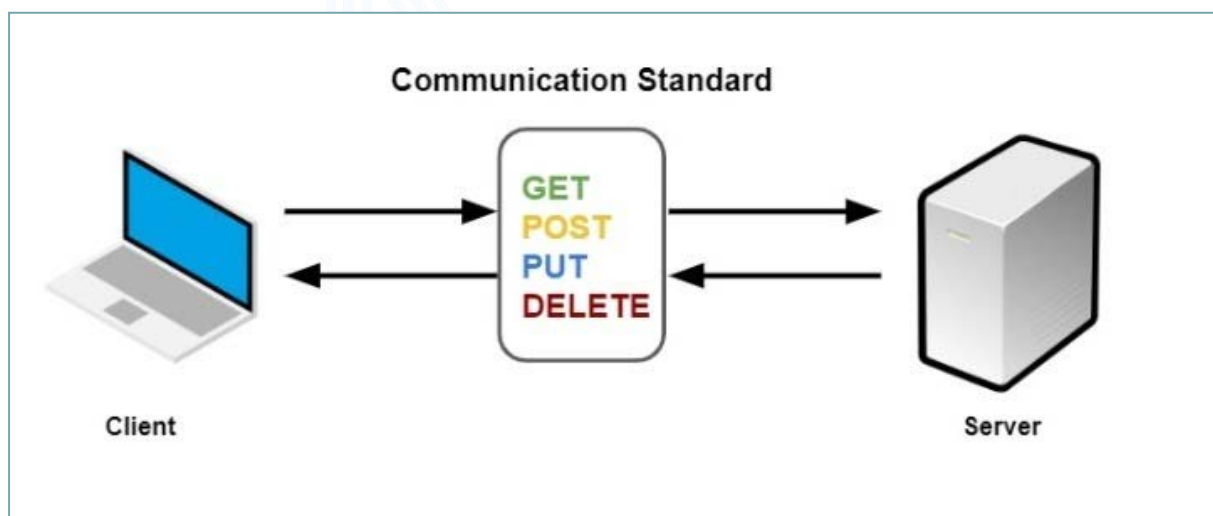


KELAS·WORK
by KELAS.COM

Membuat RESTful Service di Go-Lang

Apa itu REST?

REST merupakan singkatan *REpresentation State Transfer* yang berfungsi sebagai standar komunikasi pada web.



Manfaat REST

REST memiliki beberapa manfaat, diantaranya adalah:

- Stateless

API yang tidak memiliki *state internal* di dalam servernya dan hanya mempertimbangkan *request* yang diminta saat itu. Respon yang diberikan akan selalu sama sesuai dengan *request*.

- Cacheable

Cacheable ditujukan untuk *end point* "Get".

- Flexibility

Tidak terikat pada satu data atau *resources*. Contohnya respon dapat diberikan dalam bentuk teks, JSON, xml dan html.



KELAS·WORK
by KELAS.COM

Project

Pada project kali ini, akan membuat API service untuk sebuah restoran yang bisa melihat menu, memesan makanan dan minuman. Dalam API service restoran ini juga akan menangani *user login* dan *user sign up*.

Membuat API Pertama di Go-Lang

- Inisiasi sebuah projek baru dengan *go mod init* disertai spesifikasinya
- *Echo* adalah salah satu *framework* http di Go-Lang
- Selalu tambahkan v4 dalam penggunaan *echo*
- Setelah *go get* di run, *go mod* akan langsung terupdate dan akan muncul *go sum*
- *go sum* adalah sebuah version manager untuk melihat versi yang kita gunakan
- Buatlah *main.go* baru
- *echo.HandlerFunc* adalah fungsi yang digunakan untuk implementasi http Handler
- *Insomnia* adalah cara untuk mengakses langsung hasil koding sesuai request



Connect Service to Data Source

Jika sudah memiliki ribuan *data code*, sebaiknya gunakan *database* agar dapat di-*maintain* dan memiliki keamanan data yang baik. Di dalam proyek ini akan menggunakan PostgreSQL. PostgreSQL adalah *relational database* yang digunakan secara tabel, baris dan kolom. berikut ini cara *connect service to data source*:

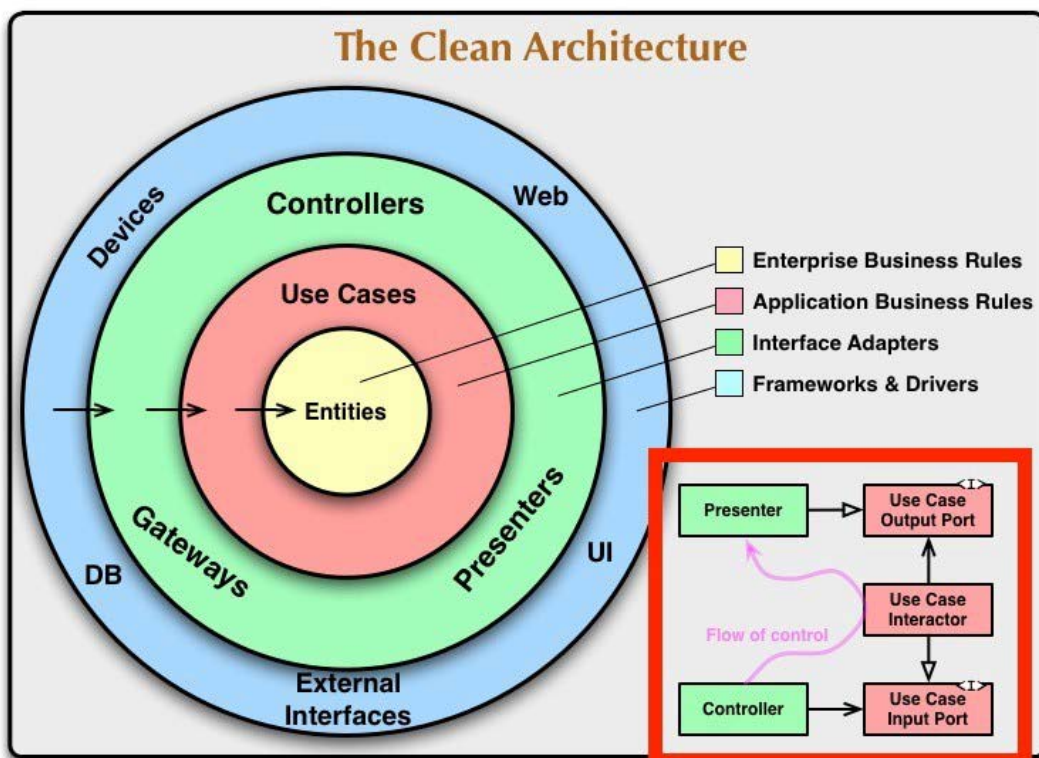
- Akses database di terminal dengan `sudo su postgres`
- Install package yang akan digunakan yaitu `go get gorm.io/gorm`
- Gorm adalah sebuah orm (*object relational model*) berfungsi untuk mengabstraksi interaksi dari koding ke database
- Install juga `go get gorm.io/driver/postgres` di terminal
- Buat spesifikasi kebutuhan di database
- Untuk menyambungkan (*connect*) service dengan data source:
 - Buka DBeaver (atau aplikasi sejenis)
 - Lalu tambahkan koneksi yaitu PostgreSQL
 - Lalu tambahkan database yang dibutuhkan





Refactoring Code to Clean Architecture

Clean architecture adalah sebuah metode untuk membagi kode menjadi beberapa bagian yang disebut sebagai "layers". Layers ini terdiri dari entity, use case, controllers, dan web (*delivery layer*).



Clean Architecture

Untuk memahami lebih jauh mengenai clean architecture dapat membaca buku berjudul *Clean Architecture: A Craftsman's Guide to Software Structure and Design* karya Robert C. Martin. Inti dari buku tersebut adalah dalam service kita harus melakukan pemisahan konsentrasi atau *separation of concerns*



Pendekatan yang dilakukan adalah dengan melakukan beberapa layer yaitu usecase, repository, dan DB yang semuanya terpisah. Kegunaan dari clean architecture adalah alur kode menjadi jelas dan terstruktur, serta tanggung jawab masing-masing bagian jadi lebih menonjol

Cara Membuat Kode Menjadi Clean Architecture

- Pindahkan *main.go* ke dalam *cmd*
- Buat juga folder *internal*
- Dalam *internal*, buatlah layer *usecase*, *repository*, *model* (sesuai kebutuhan), *delivery* dan *database*
- Membuat abstraksi database dengan *database.go*
- Import *gorm.io/gorm* dan gunakan function *GetDB*
- Tambahkan *seedDB* di *seed.go*
- *Delivery* layer dapat berbentuk *rest*
- Di dalam *rest* terdapat *handler.go* dan *router.go*
- *Repository* adalah layer akan berinteraksi dengan data source (*orm*)
- Dalam *repository* buatlah *menu.go* dan *repository.go*
- *Usecase* adalah tempat *service/domain logic* hidup, seperti proses menu adalah pengaturan order
- Dalam *usecase*, buatlah *resto.go* dan *usecase.go*
- Tambahkan file baru yaitu *menu_handler.go* dalam *delivery* layer
- Jangan simpan hal-hal yang bersifat *private* di dalam koding, sebaiknya simpan di tempat terpisah
- Pastikan kondisi semua layer sudah aman, kemudian *go run cmd/main.go* dan cek hasilnya di *Insomnia*



Membuat Order API dan Melihat Status Order

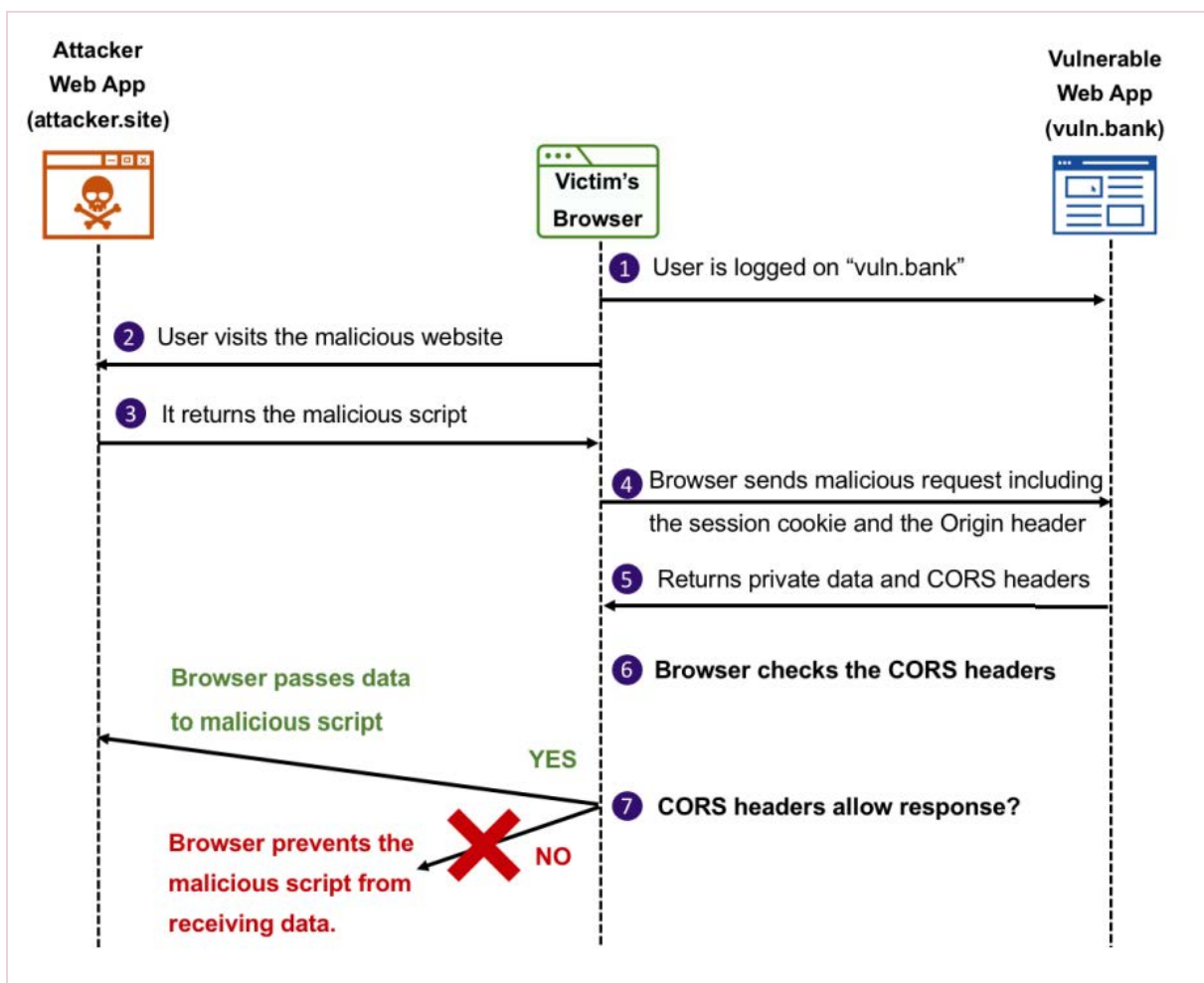
- Pilih menu "model" dan buat model baru dengan nama order.go
- Buatlah sebuah order yang memiliki Status, ProductOrders, OrderCode, Quantity dan Total Price
- Pilih menu "constant" yang pertama untuk menambahkan:
 - OrderStatusProcessed
 - OrderStatusFinished
 - OrderStatusFailed
- Masih pada menu "constant" yang kedua untuk menambahkan:
 - ProductOrderStatusPreparing
 - ProductOrderStatusFinished
- Terdapat fitur *primary key* dan *foreign key* yang berfungsi untuk mengkorelasikan satu data dengan data yang lain
- Kedua *key* ini diperlukan dalam Order dan ProductOrders
- Pilih menu "*repository*", lalu buat folder bernama order
- Buat *repository interface* yang bisa untuk CreateOrder dan GetOrderInfo
- Pilih menu "*usecase*" yang akan digunakan untuk resto.go dan usecase.go
- Buatlah *usecase interface* yang terdiri dari GetMenuList, Order dan GetOrderInfo
- Penggunaan uuid untuk ID sudah dipastikan ID akan unik (tidak sama)
- Tambahkan OrderRepo ke dalam usecase
- Setelah menambahkan Handler, jangan lupa untuk menambahkan AutoMigrate di seed.go
- Untuk menghindari *general problem* seperti pemesanan ulang dari sisi user, kita dapat menambahkan ReferenceID



Improving Security

Menambahkan CORS Middleware

Cross-Origin Resource Sharing (CORS) merupakan mekanisme browser yang memungkinkan akses terkontrol ke sumber daya yang terletak di luar domain. Sederhananya, CORS merupakan mekanisme untuk menentukan apakah sebuah *web application* di origin A diperbolehkan untuk mengakses sebuah request yang diberikan dari *resource origin* B.



Ilustrasi CORS



KELAS·WORK
by KELAS.COM

Studi Kasus CORS

Berikut ini merupakan contoh studi kasus dari CORS:

- <https://kawalcovid19.id/> adalah sebuah website yang mengambil data dari berbagai sumber seperti cekdiri.id



- Cekdiri.id mengimplementasikan CORS pada keamanan webnya, sehingga tidak memperbolehkan kawalcovid19.id untuk mengakses resource-nya



```
✖ Access to fetch at 'https://cekdiri.id/vaksinasi/' from origin vaksin:1
'https://kawalcovid19.id' has been blocked by CORS policy: No 'Access-
Control-Allow-Origin' header is present on the requested resource. If an
opaque response serves your needs, set the request's mode to 'no-cors'
to fetch the resource with CORS disabled.

✖ ▶ GET https://cekdiri.id/vaksinasi/ net::ERR_FAILED 301 fetch.ts:15
✖ Access to fetch at 'https://cekdiri.id/vaksinasi/' from origin vaksin:1
'https://kawalcovid19.id' has been blocked by CORS policy: No 'Access-
Control-Allow-Origin' header is present on the requested resource. If an
opaque response serves your needs, set the request's mode to 'no-cors'
to fetch the resource with CORS disabled.

✖ ▶ GET https://cekdiri.id/vaksinasi/ net::ERR_FAILED 301 fetch.ts:15
```

- Maka, browser tidak dapat memberikan hasilnya

Situasi Vaksinasi COVID-19 Saat Ini

Lihat Lebih Lengkap →

Vaksinasi Dosis 1



Vaksinasi Dosis 2



Error - Gagal mengambil data terbaru
Gagal mengambil data. Mohon coba lagi dalam beberapa saat.



Implementasi CORS Middleware

Middleware adalah sebuah fungsi yang dijalankan sebelum masuk ke *logic handler* http. Cara untuk *install middleware* adalah sebagai berikut:

- Menggunakan `go get github.com/labstack/echo/v4/middleware` di terminal
- Tambahkan *middleware.go* di *rest*
- Buka Insomnia (request http tool)



KELAS·WORK
by KELAS.COM

Membuat Sign Up Flow dengan Argon2

Apa itu JWT?

Seorang backend harus membuat *user management* supaya user dapat *register* dan *login* di API service web milik sendiri. Cara untuk mengetahui kebenaran user adalah dengan menggunakan akses token JWT. JWT (JSON Web Token) merupakan sebuah standar untuk mengirimkan informasi antar dua pihak menggunakan JSON *object* secara aman.

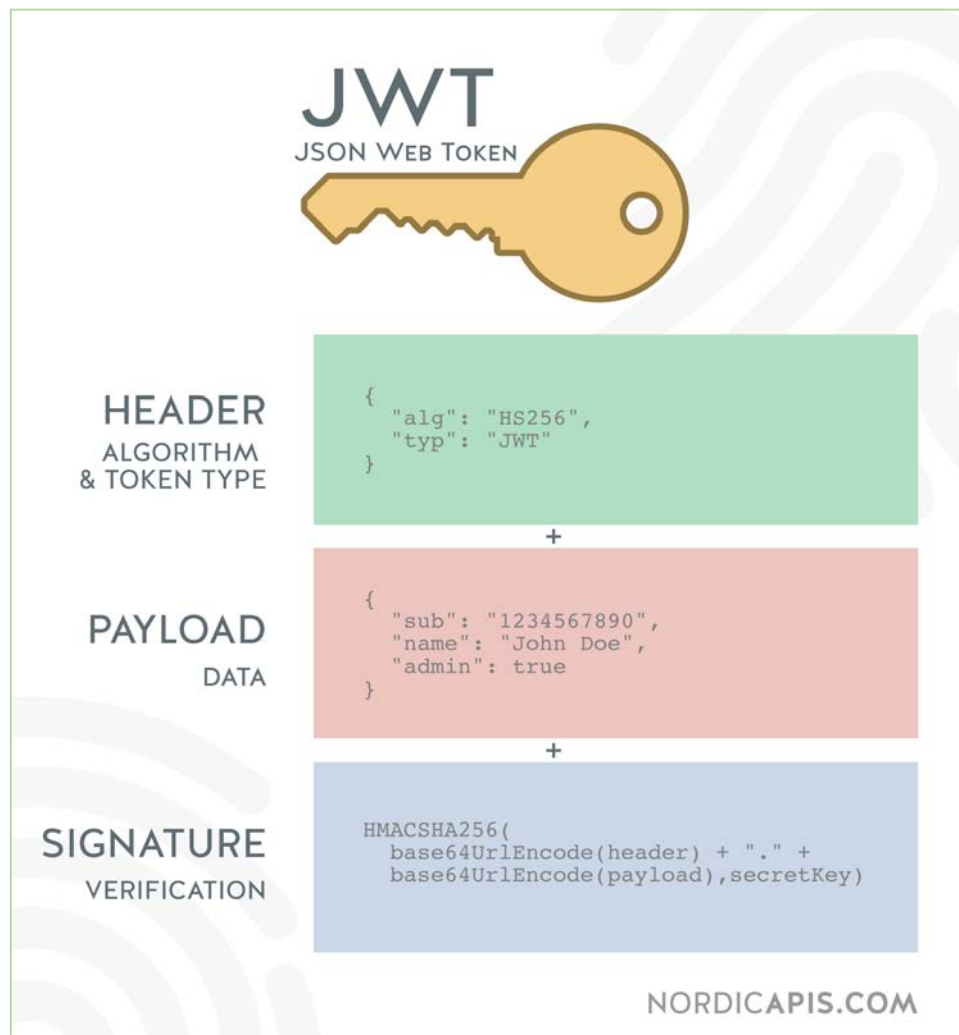
Struktur JWT

JWT terbagi menjadi beberapa struktur, yaitu:

- *Header* terbagi dua hal yaitu tipe token dan algoritma
- *Payload* berisi data yang ingin diberikan ke pihak lain (*claims*)
- *Claim*: *registered*, *public*, dan *private claims*
- *Signature* diambil dengan cara mengambil data dari *header* dan *payload*, dan di "*sign*" dengan algoritma *signing* yang sudah ditentukan



KELAS·WORK
by KELAS.COM



Implementasi Sign Up Flow dengan Argon2

- Buat model user dengan cara membuat folder *user.go* di *model*
- Sebaiknya tidak menyimpan password user di *database*, tapi hanya simpan *hash password*-nya saja
- Membuat RegisterRequest dengan memunculkan *username* dan *hash password*
- Argon2 adalah sebuah algoritma *hash* yang keluar di tahun 2015 dan terbilang cukup baru



- Argon2 kini menjadi rekomendasi algoritma *hashing* untuk *password*
- Argon2 sudah termasuk *build-in package* dari Go-Lang
- Buatlah *repository.go* di menu *user*
- Kemudian buatlah *Repository interface* untuk membuat *RegisterUser*, *CheckRegistered* dan *GenerateUserHash*
- Untuk membuat *GenerateUserHash*, buatlah *hash.go* dalam menu *user* yang terdapat dalam *repository*
- Argon2 membutuhkan *IDKey*, *salt*, *time*, *memory*, *threads* dan *keylen*
- Dalam *Usecase interface* tambahkan *RegisterUser*
- *NewChiper* sudah termasuk *build-in* dari Go-Lang
- Setelah koding selesai lakukan *go run cmd/main.go*
- Lalu lakukan tes request di aplikasi Insomnia
- Cek juga DB-nya di aplikasi DBeaver

Membuat Login Session dengan JWT

- Masuk ke dalam menu *repository* dan tambahkan *VerifyLogin* pada *Repository interface*
- Tambahkan juga *GetUserData* dan *CreateUserSession* pada *Repository interface*
- Buka menu *user.go* dan tambahkan *UserSession* menggunakan *JWTToken*
- Untuk implementasi *UserSession* install *go get github.com/golang-jwt/jwt*
- Tambahkan *session.go* dalam package *user* dalam *repository*
- Buatlah *accessToken* dari *UserID*
- Buatlah *Claims* dalam *session.go*
- Dalam *accessToken* hanya menggunakan *jwt.StandardClaims* yaitu: *Subject* dan *ExpiresAt*



- Tambahkan Login pada Usecase interface dalam *usecase.go*
- Cek *login session* dalam aplikasi Insomnia dan kita akan mendapatkan jwt token
- Untuk mengecek isi jwt token diperlukan *website* yaitu *jwt.io* yang berfungsi untuk cek *debugging*

Menambahkan AUTH Middleware ke API

- Tambahkan CheckSession di Usecase interface pada package *usecase.go*
- Buatlah authMiddleware pada package *middleware.go*
- Buatlah *utils.go* pada package *rest* dan buat function GetSessionData
- authContext adalah sebuah konteks dalam suatu perjalanan request
- Cek hasil kode dengan membuat order baru di aplikasi Insomnia





Improving Security Reliability

Implementasi Logging di Service

Agar dapat meningkatkan *security reliability*, dapat menggunakan *observability*. Cara meningkatkan *observability* adalah dengan menambahkan *logging*. Go-Lang sudah memiliki *built in packed* untuk *logging*. Untuk itu, cara mengimplementasikan *logging* di service adalah sebagai berikut:

- Projek ini akan menggunakan logrus yang bisa di-install melalui `go get github.com/sirupsen/logrus`
- Langkah pertama yaitu membuat package *logger* pada internal Go-Lang dan buat menu *logger.go*
- Membuat function *Init*, lalu buat *logrus.SetOutput*, *logrus.SetLevel*
- Setelah kode selesai cek menggunakan aplikasi Insomnia

Hal Penting dalam yang harus diperhatikan pada *logging*:

- Gunakan *logging* hanya untuk hal penting saja
- Jangan memasukkan data-data sensitif seperti *password*
- Simpan dan proses *logging* di tempat lain

Implementasi Open Telemetry

Open telemetry menyediakan *open structure data* untuk observabilitas mulai dari *logging*, *metrics* dan *traces*, lengkap dengan *tools*-nya. Berikut ini cara implementasi *open telemetry*:

- Instal *open telemetry* pada `go get go.opentelemetry.io/otel` dan harus mengambil `go.opentelemetry.io/otel/trace`



- Pada *open telemetry* kita akan menggunakan *context* (seperti pada pembahasan *middle.ware*) untuk menyimpan *data tracing*
- Dalam langkah pembuatan *wrapper*, buat folder *tracing* dan *tracing.go* pada package *menu*
- Buatlah function *createTracingProvider*
- Instal *go.opentelemetry.io/otel/sdk/trace*
- Instal *go.opentelemetry.io/exporters/jaegers*
- Cek hasil koding *tracing* pada *Docker* (instal di *docker run -d -name jaeger*)
- menggunakan format *-p 14268: 14268 *
- Setelah berhasil di *Docker run*, kembali cek *go run cmd/main.go* di VS Code
- Cek laman *jaeger* di *localhost:16686/search*
- Setelah selesai kemudian lakukan tes hit di aplikasi *Insomnia*
- Setelah *request* di aplikasi *Insomnia* selesai, kembali *refresh* laman *jaeger* untuk mengecek (*trace*) *request* yang masuk

Handling Panics

Panic menghentikan program di Go-Lang untuk melanjutkan apa yang seharusnya dilakukan. Untuk membuat *service* yang *reliable*, perlu memprogram *panic* dan tidak mengganggu program yang lain.

Implementasi Unit Test

- Sebagai contoh pembuatan *unit tests*, ambil salah satu *usecase* yaitu *GetMenuList*
- Gunakan *Go Mock* dan instal di *go install*



github.com/goalng/mock/mockgen@v1.6.0

- Cara manual untuk *generate mock*: *mockgen -package=mocks -mock_names=Usecase=MockRestoUsecase -destination=.internalmocks/resto_usecase,mock.go -source=.internal/usecase/resto/usecase.go*
- Setelah kita *run* akan muncul folder baru yaitu *mock* yang di dalamnya ada *resto_usecase_mock.go* yang sudah memiliki banyak implementasi
- Pada terminal lakukan *go get github.com/golang/mock/gomock*
- Cara kedua untuk *generate mock*: masuk ke *repository.go* dan lakukan *go generate mockgen -package=mocks -mock_names=Repository=MockMenuRepository -destinention=../mocks/menu_repository_mock.go -source=repository.go*
- Lalu pada terminal lakukan *go generate ./...*
- Untuk pembuatan unit test, klik *resto.go*, klik kanan kemudian pilih *Go: Generate Unit Tests for Function* dan akan otomatis ada folder *resto_test.go*
- Instal *go get github.com/DATA-DOG/go-sqlmock* pada terminal

Behavior Driven Development Using Ginkgo

BDD (*Behavior Driven Development*) merupakan sebuah *framework* yang memfokuskan *testing* pada sebuah kumpulan perilaku yang dijelaskan dalam bahasa manusia. BDD berfungsi sebagai jembatan antara tim *engineer* dan tim bisnis. Salah satu *library* yang support BDD adalah Ginkgo. Berikut ini cara menginstal Ginkgo:



- Cara instal Ginkgo pada terminal: `go install github.com/onsi/ginkgo/v2/ginkgo`
- Kemudian `go get github.com/onsi/ginkgo/v2/ginkgo`
- Lalu `go get github.com/onsi/gomega/...`
- `cd internal/usecase/resto`
- Kemudian run `ginkgo bootstrap`
- Setiap *test case behaviour scenario* dalam Ginkgo akan digunakan `BeforeEach`
- Penggunaan unit test dengan Ginkgo atau unit test dari Go-Lang bisa menjadi preferensi masing-masing

TDD vs BDD

TDD dan BDD memiliki perbedaan baik dalam fungsi dan tujuannya. Namun TDD dan BDD sebaiknya digunakan secara bersamaan. Untuk menggunakan kedua fungsi ini sebaiknya membuat terlebih dahulu sebuah *behavior* yang diinginkan dari sebuah fitur, lalu menggunakan test development untuk iterasi yang dapat memenuhi keperluan tersebut.

Test-Driven Development	Behavior-Driven Development
Sinergi keseluruhan fungsi dari komponen yang ada	Lebih berfokus pada fungsionalitas masing-masing komponen
Bisnis menggunakan TDD memiliki banyak tes yang bisa jadi mencapai nilai bisnis atau tidak bisa mencapai nilai bisnis	Dalam bisnis, BDD memungkinkan untuk melihat kepada nilai bisnis lalu menelusuri rangkaian fitur
Programmer mendapatkan <i>feedback</i> dari kode	Tim mendapatkan <i>feedback</i> dari pemilik produk



Advanced Go-Lang

Concurrency

Goroutines merupakan fitur untuk menjalankan lebih dari satu fungsi secara bersamaan dalam Go-lang. Goroutines berbeda dengan *threading*. Goroutines lebih ringan dan efisien dibandingkan dengan *multithreading* biasa. Berikut cara untuk menggunakan Goroutines:

- Menggunakan Goroutines dapat menuliskan kata Go dalam fungsi yang akan dipanggil
- Jika *output* tidak muncul ketika dijalankan, maka dapat menambahkan `time.Sleep`
- `Time.Sleep` sebenarnya tidak baik digunakan untuk *service* karena akan menggunakan *wait group*
- Untuk menggunakan *wait group* dapat menuliskan `wg sync.WaitGroup`
- Mengakses *resource* secara bersamaan dengan aman
- *Data Race*: Ketika *previous value* dan *current value* berubah di tengah program dan tidak ada proteksi
- Menyelesaikan masalah *data race* menggunakan *mutual explosion* atau Mutex
- Mutex akan menyebabkan Goroutines untuk menunggu sampai *resource* dapat dipakai

Channels

Channels merupakan cara Goroutines yang berbeda-beda untuk berkomunikasi satu dengan lainnya. Untuk menginisiasi *channel* dapat menggunakan huruf `c`.



tahapan yang dapat dilakukan untuk menggunakan *Channel* adalah sebagai berikut:

- Menginisiasi *channels* dengan C (huruf c)
- Mengambil *value* dalam *channels* menggunakan simbol <- (panah ke kiri)
- *Channels* dapat digunakan untuk menunggu sesuatu sampai selesai
- *Channels* mempunyai cara khusus untuk *conditionals* dengan menggunakan *select*
- *Channels* dapat membedakan mana *channels* yang terbuka dan yang tidak dengan menambahkan satu *variable* saat menerima data *channels* tersebut
- Menutup *channels* menggunakan *close*

Generics

Generic dikenal sebagai fitur baru di Go-lang yang muncul pada versi 1.18. *Generic* dapat digunakan untuk membuat fungsi yang bisa berinteraksi dengan lebih dari satu tipe dan menjumlahkan data-data yang memiliki tipe data yang berbeda dengan *maintain* fungsi-fungsi yang sama namun tipe data yang berbeda.

Implementasi Generics

- *Generics* dapat menggabungkan beberapa fungsi menjadi hanya satu fungsi saja
- *Generics* dipanggil dengan cara memberikan tipe data sebelum mengisi fungsi seperti biasanya
- Menggunakan *interface* untuk membuat fungsi yang lebih *generics*
- *Generics* dapat meminimalisir penggunaan *code* yang tidak perlu



Caching

Caching merupakan sebuah tempat penyimpanan data yang berfokus pada performa. *Cache* ditujukan sebagai tempat penyimpanan data sementara. Kebanyakan sistem *caching* menggunakan *random access memory* untuk menyimpan data.

Implementasi Caching

- Ketika sudah mendapatkan data maka langsung simpan ke dalam *cache*
- *Cache* dapat melakukan dua hal, *set* dan *get*
- *Cache* dapat digunakan untuk menghemat waktu
- *Library cache* umumnya menerapkan strategi untuk menghapus data dalam *cache* melalui *expired time*
- Data di dalam *cache* juga dapat dihapus secara manual