

Digitalent Kominfo - Implementasi MVC Golang

Modul hands-on ini memfasilitasi pembentukan kompetensi untuk mampu melakukan implementasi MVC menggunakan Golang. Target dari modul ini adalah peserta mampu membuat sistem perbankan sederhana dan dapat terintegrasi dengan database menggunakan arsitektur MVC. Kode lengkap dari modul pelatihan ini dapat dilihat melalui link berikut https://github.com/FadhlanHawali/Digitalent-Kominfo_Implementation-MVC-Golang .

Bagian I

1. Menginstall Package Gorm ORM

Gorm merupakan sebuah ORM atau Object Relational Mapping dimana ORM mempermudah developer untuk melakukan proses query pada database tanpa perlu menuliskan “Raw Query” untuk melakukan proses query.

```
go get -u gorm.io/gorm
```

2. Menginstall Package Gorm ORM MySQL Support

```
go get -u gorm.io/driver/mysql
```

3. Menginstall Package Gin

```
go get -u github.com/gin-gonic/gin
```

4. Menginstall Package Gin Cors

```
go get -u github.com/gin-contrib/cors
```

5. Struktur Folder MVC

```
| -- main.go          #menjadi gerbang utama dari main app
| -- views            #folder yang nantinya akan berisi view berupa html
|   |-- index.html
| -- assets           #akan berisi aset-aset seperti gambar jika dibutuhkan
|   |-- ...
| -- app
|   |-- config
|     |-- config.go
|   |-- controller
|     |-- controller.go
|     |-- db.go
|   |-- handler
|     |-- handler.go
|   |-- middleware
|   |-- model
|     |-- bank.go
|   |-- utils
|   |-- wrapper.go
```

6. Setting Config DB

Proses ini berfungsi untuk melakukan setup awal koneksi database MySQL. Jika belum memiliki MySQL, maka peserta dapat melakukan install MySQL terlebih dahulu.

Windows

Unduh melalui <https://dev.mysql.com/downloads/installer/>, untuk melihat detail instruksi untuk install, dapat kunjungi link berikut
<https://dev.mysql.com/doc/refman/8.0/en/windows-installation.html#:~:text=The%20simplest%20and%20recommended%20method,%2Finstaller%2F%20and%20execute%20it>.

Linux

Dapat mengunjungi link berikut
<https://www.digitalocean.com/community/tutorials/how-to-install-mysql-on-ubuntu-18-04>

Jika sudah terinstall, maka proses selanjutnya adalah insialisasi koneksi ke MySQL pada file /config/config.go

```

package config

import (
    "fmt"
    "github.com/FadhlanHawali/Digitalent-Kominfo_Implementation-MVC-Golang/app/model"
    "gorm.io/driver/mysql"
    "gorm.io/gorm"
)

func DBInit() *gorm.DB{
    db, err := gorm.Open(mysql.Open(fmt.Sprintf("root:pintar123@digitalent_bank?
charset=utf8&parseTime=True&loc=Local")), &gorm.Config{})
    if err != nil {
        panic("failed to connect to database" + err.Error())
    }
    db.AutoMigrate(new(model.Account), new(model.Transaction))
    return db
}

```

Kemudian kita menambahkan entity DB pada file controller/db.go.

```

package controller

import "gorm.io/gorm"

type InDB struct {
    DB *gorm.DB
}

```

Proses selanjutnya yaitu merancang model dari data yang nantinya akan disimpan kedalam database. Tambahkan model pada file model/bank.go seperti berikut

```

type Account struct {
    ID int `gorm:"primary_key" json:"-`
    IdAccount string `json:"id_account,omitempty"`
    Name string `json:"name"`
    Password string `json:"password,omitempty"`
    AccountNumber int `json:"account_number,omitempty"`
    Saldo int `json:"saldo"`
}

type Auth struct {
    Name string `json:"name"`
    Password string `json:"password"`
}

type Transaction struct {
    ID int `gorm:"primary_key" json:"-`
    TransactionType int `json:"transaction_type,omitempty"`
    TransactionDescription string `json:"transaction_description"`
    Sender int `json:"sender"`
    Amount int `json:"amount"`
    Recipient int `json:"recipient"`
    Timestamp int64 `json:"timestamp,omitempty"`
}

```

7. Membuat Skeleton Controller

Proses ini menambahkan controller handler yang belum memiliki logic pada folder controller/handler.go. Pada folder ini nantinya akan berisi handler seperti pada pelatihan sebelumnya.

```
package controller

import "github.com/gin-gonic/gin"

func (inDB *InDB) CreateAccount(c *gin.Context){

}
```

8. Finalisasi Fungsi Main

Insialisasi Gin router pada main.go dan inisialisasi koneksi database dengan menambahkan kode berikut.

```
package main

import (
    "github.com/FadhlanHawali/Digitalent-Kominfo_Implementation-MVC-Golang/app/config"
    "github.com/FadhlanHawali/Digitalent-Kominfo_Implementation-MVC-Golang/app/controller"
    "github.com/gin-contrib/cors"
    "github.com/gin-gonic/gin"
)

func main(){
    db := config.DBInit()
    inDB := &controller.InDB{DB: db}

    router := gin.Default()
    router.Use(cors.Default())
    router.GET("/", inDB.CreateAccount)
    router.Run(":8080")
}
```

9. Jalankan program

Jika seluruh kode sudah ditambahkan, maka program dapat dijalankan dengan memasukkan command berikut pada terminal

Linux

```
go build
./Digitalent-Kominfo_Implementation-MVC-Golang
```

Windows

```
go build
Digitalent-Kominfo_Implementation-MVC-Golang.exe
```

Untuk melihat apakah koneksi dari backend ke database berhasil atau belum ditunjukkan dengan melihat pada console MySQL apakah tabel sudah terbentuk atau belum seperti berikut

```
mysql> show tables;
+-----+
| Tables_in_digitalent_bank |
+-----+
| accounts                  |
| transactions              |
+-----+
2 rows in set (0.00 sec)
```

Bagian II

Pada bagian II ini kita akan melengkapi seluruh handler dan menambahkan proses logic dari transaksi database yang sudah dirancang sebelumnya.

1. Melengkapi Utils/Helper

File utils ini nantinya berfungsi untuk helper dalam menyelesaikan logic yang dibutuhkan dalam proses transaksi. Terdapat 3 helper yaitu :

a. utils/bcrypt.go

Pada file ini berisi proses logic untuk menghasilkan hash password dalam membuat akun. Sebelum menggunakan bcrypt ini, pengguna diharuskan menginstall package bcrypt terlebih dahulu dengan menuliskan command seperti berikut :

```
go get golang.org/x/crypto/bcrypt
```

Terdapat 2 fungsi utama yaitu HashGenerator dan HashComparator.

HashGenerator sendiri berfungsi untuk menghasilkan hash dari sebuah string password yang dimasukkan saat membuat akun. Hal ini sangat penting karena menambah tingkat keamanan dari akun itu sendiri. Bentuk password yang sudah dihash adalah seperti pada gambar dibawah.

name	password	account_number	saldo
fadhlan	\$2a\$10\$vmSnSzBdX0A1w/V9JAo.9.TzE8NwBvvW3CXTMy3djsoCtXS52s84.	6789	0
fadhaw	\$2a\$10\$a/5gwpB6jvoaiul/DQZGAegEFev/CsEqYRXLLIyiaD5yLL.0Lk4RS	832712	10500
fad123	\$2a\$10\$5sR./DgEPkXPXUQvDLH.2.voB0AE7afiqAMqn4DPaNmXaZrQr3DFK	577718	500

Kemudian untuk HashComparator sendiri berfungsi untuk komparasi apakah password yang dimasukkan saat login memiliki "susunan" / struktur yang sama dengan hash yang sudah disimpan sebelumnya.

```

package utils

import "golang.org/x/crypto/bcrypt"

func HashGenerator(str string) (string, error) {
    hashedString, err := bcrypt.GenerateFromPassword([]byte(str), bcrypt.DefaultCost); if err != nil {
        return "", err
    }

    return string(hashedString), nil
}

func HashComparator(hashedByte []byte, byte []byte) (error){
    err := bcrypt.CompareHashAndPassword(hashedByte, byte); if err != nil {
        return err
    }

    return nil
}

```

b. `utils/Idgenerator.go`

Pada file ini berisi helper untuk menghasilkan random number dengan range yang diinginkan.

```

package utils

import "math/rand"

func RangeIn(low, hi int) int {
    return low + rand.Intn(hi-low)
}

```

c. `utils/wrapper.go`

Pada file ini sebenarnya hanya untuk mempermudah kita saat ingin mengeluarkan output, sehingga kita tidak perlu menulis c.JSON berulang kali jika ingin mengeluarkan output pada backend. Terdapat 3 wrapper yaitu:

- WrapAPIError
Wrapper ini dipanggil jika keluaran yang diinginkan berupa body yang berisi error
- WrapAPISuccess
Wrapper ini dipanggil jika keluaran yang diinginkan berupa success message saja
- WrapAPIData
Wrapper ini dipanggil jika keluaran yang diinginkan berupa success message dan data output.

```

package utils

import (
    "github.com/gin-gonic/gin"
    "net/http"
)

func WrapAPIError(c *gin.Context, message string, code int) {
    c.JSON(code, map[string]interface{}{
        "code":      code,
        "error_type": http.StatusText(code),
        "error_details": message,
    })
}

func WrapAPISuccess(c *gin.Context, message string, code int) {
    c.JSON(code, map[string]interface{}{
        "code":      code,
        "status":     message,
    })
}

func WrapAPIData(c *gin.Context, data interface{}, code int, message string) {
    c.JSON(code, map[string]interface{}{
        "code":      code,
        "status":     message,
        "data":      data,
    })
}

```

2. Melengkapi Model

Pada model ini nantinya berisi logic yang berkaitan dengan transaksi dengan database.

File ini terdapat di model/bank.go. Terdapat 6 logic utama dalam aplikasi ini :

a. Login

Pada logic ini bertugas untuk mengurus hal autentikasi saat pengguna melakukan login dengan melakukan komparasi pada data yang terdapat pada database. Jika berhasil maka akan mengeluarkan hasil berupa JWT Token yang nantinya dapat digunakan untuk proses pemanggilan API lainnya.

```

func Login(auth Auth) (bool, error, string){
    var account Account
    if err := DB.Where(&Account{Name: auth.Name}).First(&account).Error;
    err!=nil{
        if err == gorm.ErrRecordNotFound{
            return false, errors.Errorf("Account not found"), ""
        }
    }

    err := utils.HashComparator([]byte(account.Password), []byte(auth.Password))
    if err != nil{
        return false, errors.Errorf("Incorrect Password"), ""
    } else {

        sign := jwt.NewWithClaims(jwt.SigningMethodHS256, jwt.MapClaims{
            "name": auth.Name,
            "account_number": account.AccountNumber,
        })

        token, err := sign.SignedString([]byte("secret"))
        if err != nil {
            return false, err, ""
        }
        return true, nil, token
    }
}

```

b. InsertNewAccount

Pada proses ini berfungsi untuk menambahkan akun yang ingin didaftarkan kedalam database.

```
func InsertNewAccount(account Account) (bool,error){
    account.AccountNumber = utils.RangeIn(111111,999999)
    account.Saldo = 0
    account.IdAccount = fmt.Sprintf("id-%d",utils.RangeIn(111,999))
    if err := DB.Create(&account).Error;err!=nil{
        return false, errors.Errorf("invalid prepare statement :%+v\n", err)
    }
    return true,nil
}
```

c. GetAccountDetail

Proses ini berfungsi untuk mendapatkan detail akun dan daftar transaksi yang pernah dilakukan.

```
func GetAccountDetail(idAccount int) (bool,error, []Transaction,Account){
    var transaction []Transaction
    var account Account
    if err := DB.Where("sender = ? OR recipient = ?",idAccount,idAccount).
        Find(&transaction).Error;err!=nil{
        if err == gorm.ErrRecordNotFound{
            return false,errors.Errorf("Account not found"),
            []Transaction{},Account{}
        } else {
            return false, errors.Errorf("invalid prepare statement :%+v\n", err),
            []Transaction{},Account{}
        }
    }

    if err := DB.Where(&Account{AccountNumber:
        idAccount}).Find(&account).Error;err != nil{
        if err == gorm.ErrRecordNotFound{
            return false,errors.Errorf("Account not found"),
            []Transaction{},Account{}
        } else {
            return false, errors.Errorf("invalid prepare statement :%+v\n", err),
            []Transaction{},Account{}
        }
    }

    return true,nil,transaction,Account{
        IdAccount:    account.IdAccount,
        Name:         account.Name,
        AccountNumber: account.AccountNumber,
        Saldo:        account.Saldo,
    }
}
```

d. Transfer

Pada transfer ini berfungsi untuk melakukan transfer uang kepada pengguna lain.


```

func Transfer (transaction Transaction) (bool,error){
    err := DB.Transaction(func(tx *gorm.DB) error {
        // do some database operations in the transaction (use 'tx' from this
        point, not 'db')
        var sender,recipient Account
        if err := tx.Model(&Account{}).Where(&Account{AccountNumber:
transaction.Sender}).
            First(&sender).
            Update("saldo", sender.Saldo-transaction.Amount).Error; err != nil {
            // return any error will rollback
            return err
        }
        if err := tx.Model(&Account{}).Where(&Account{AccountNumber:
transaction.Recipient}).
            First(&recipient).
            Update("saldo", recipient.Saldo+transaction.Amount).Error; err != nil
        {
            // return any error will rollback
            log.Println("ERROR : " + err.Error())
            return err
        }
        transaction.TransactionType = constant.TRANSFER
        transaction.Timestamp = time.Now().Unix()
        if err := tx.Create(&transaction).Error;err != nil {
            return err
        }
        // return nil will commit the whole transaction
        return nil
    });if err != nil {
        return false, err
    }
    return true,nil
}

```

e. Withdraw

Pada fungsi ini berfungsi untuk melakukan penarikan dari saldo yang dimiliki oleh pengguna.

```

func Withdraw (transaction Transaction) (bool,error){
    err := DB.Transaction(func(tx *gorm.DB) error {
        var sender Account
        if err := tx.Model(&Account{}).Where(&Account{AccountNumber:
transaction.Sender}).
            First(&sender).
            Update("saldo", sender.Saldo-transaction.Amount).Error; err != nil {
            // return any error will rollback
            return err
        }
        transaction.TransactionType = constant.WITHDRAW
        transaction.Timestamp = time.Now().Unix()
        if err := tx.Create(&transaction).Error;err != nil {
            return err
        }
        return nil
    });if err != nil {
        return false, err
    }
    return true,nil
}

```

f. Deposit

Fungsi deposit berguna untuk simulasi menambahkan uang ke akun pengguna.

```
func Deposit (transaction Transaction) (bool,error){
    err := DB.Transaction(func(tx *gorm.DB) error {
        var sender Account
        if err := tx.Model(&Account{}).Where(&Account{AccountNumber:
transaction.Sender}).
            First(&sender).
            Update("saldo", sender.Saldo+transaction.Amount).Error; err != nil {
            // return any error will rollback
            return err
        }
        transaction.TransactionType = constant.DEPOSIT
        transaction.Timestamp = time.Now().Unix()
        if err := tx.Create(&transaction).Error;err != nil {
            // return any error will rollback
            return err
        }
        return nil
    });if err != nil {
        return false, err
    }

    return true,nil
}
```

3. Menambahkan Middleware

Middleware ini berfungsi untuk mencegah agar tidak bisa sembarang orang mengakses API yang telah dibuat sebelumnya, sehingga hanya orang-orang yang telah memiliki akun saja yang dapat mengakses API yang telah dibuat. Proses middleware auth ini menggunakan JWT Token. Untuk selengkapnya mengenai apa itu JWT Token dapat mengunjungi link berikut <https://jwt.io/introduction/>. Terdapat beberapa package yang harus diinstal terlebih dahulu yaitu :

- JWT Go

Merupakan salah satu library standar JWT untuk Golang

```
go get github.com/dgrijalva/jwt-go
```

- Mapstructure

Library ini berfungsi untuk men-decode data dari JWT Token.

```
go get github.com/mitchellh/mapstructure
```

```
package middleware

import (
    "fmt"
    "github.com/dgrijalva/jwt-go"
    "github.com/gin-gonic/gin"
    "github.com/mitchellh/mapstructure"
    "net/http"
)

func Auth(c *gin.Context) {
    tokenString := c.Request.Header.Get("Authorization")
    token, err := jwt.Parse(tokenString, func(token *jwt.Token) (interface{}, error) {
        if jwt.GetSigningMethod("HS256") != token.Method {
            return nil, fmt.Errorf("Unexpected signing method: %v", token.Header["alg"])
        }
        return []byte("secret"), nil
    })

    if token != nil && err == nil {
        fmt.Println("token verified")
        claims := token.Claims.(jwt.MapClaims)
        fmt.Println(claims)
        var idAccount int
        err := mapstructure.Decode(claims["account_number"], &idAccount); if err != nil {
            result := gin.H{
                "message": err.Error(),
            }
            c.JSON(http.StatusUnauthorized, result)
            c.Abort()
        }
        c.Set("account_number", idAccount)
    } else {
        result := gin.H{
            "message": "token tidak valid",
            "error":   err.Error(),
        }
        c.JSON(http.StatusUnauthorized, result)
        c.Abort()
    }
}
```

4. Melengkapi Handler

Terdapat 6 Handler yang dibuat pada file controller/handler.go yaitu sebagai berikut :


a. CreateAccount

```
func CreateAccount (c *gin.Context){  
  
    var account model.Account  
    if err := c.Bind(&account); err!= nil {  
        utils.WrapAPIError(c,err.Error(),http.StatusBadRequest)  
        return  
    }  
    pass, err := utils.HashGenerator(account.Password); if err != nil{  
        utils.WrapAPIError(c,err.Error(),http.StatusBadRequest)  
        return  
    }  
    account.Password = pass  
    flag,err := model.InsertNewAccount(account)  
    if flag{  
        utils.WrapAPISuccess(c,"success",http.StatusOK)  
        return  
    }else {  
        utils.WrapAPIError(c,err.Error(),http.StatusBadRequest)  
        return  
    }  
}
```

b. GetAccount

```
func GetAccount (c *gin.Context){  
    idAccount := c.MustGet("account_number").(int)  
    flag,err,trx,acc := model.GetAccountDetail(idAccount);if err != nil {  
        utils.WrapAPIError(c,err.Error(),http.StatusInternalServerError)  
        return  
    }  
    if flag{  
        utils.WrapAPIData(c,map[string]interface{}){  
            "account":acc,  
            "transaction":trx,  
        },http.StatusOK,"success")  
        return  
    }  
}
```


c. Transfer



```
func Transfer (c *gin.Context){
    var transaction model.Transaction
    if err := c.Bind(&transaction); err!= nil {
        utils.WrapAPIError(c,err.Error(),http.StatusBadRequest)
        return
    }

    flag,err := model.Transfer(transaction); if flag{
        utils.WrapAPISuccess(c,"success",http.StatusOK)
        return
    }else {
        utils.WrapAPIError(c,err.Error(),http.StatusBadRequest)
        return
    }
}
```


d. Withdraw



```
func Withdraw (c *gin.Context){
    var transaction model.Transaction
    if err := c.Bind(&transaction); err!= nil {
        utils.WrapAPIError(c,err.Error(),http.StatusBadRequest)
        return
    }

    flag,err := model.Withdraw(transaction); if flag{
        utils.WrapAPISuccess(c,"success",http.StatusOK)
        return
    }else {
        utils.WrapAPIError(c,err.Error(),http.StatusBadRequest)
        return
    }
}
```

e. Deposit



```
func Deposit (c *gin.Context){
    var transaction model.Transaction
    if err := c.Bind(&transaction); err!= nil {
        utils.WrapAPIError(c,err.Error(),http.StatusBadRequest)
        return
    }

    flag,err := model.Deposit(transaction); if flag{
        utils.WrapAPISuccess(c,"success",http.StatusOK)
        return
    }else {
        utils.WrapAPIError(c,err.Error(),http.StatusBadRequest)
        return
    }
}
```

f. Login

```

func Login(c *gin.Context){
    var auth model.Auth
    if err := c.Bind(&auth); err != nil {
        utils.WrapAPIError(c,err.Error(),http.StatusBadRequest)
        return
    }
    log.Println("LOGIN")
    flag,err,token := model.Login(auth); if flag{
        utils.WrapAPIData(c,map[string]interface{}{
            "token":token,
        },http.StatusOK,"success")
    }else {
        utils.WrapAPIError(c,err.Error(),http.StatusBadRequest)
    }
}

```

5. Menambahkan Constant

Constant disini hanya berperan sebagai helper untuk data konstan. Terdapat 3 data konstan yaitu TRANSFER bernilai 0, WITHDRAW bernilai 1, dan DEPOSIT bernilai 2.

```

package constant

const (
    TRANSFER = 0
    WITHDRAW = 1
    DEPOSIT  = 2
)

```

6. Mengupdate main.go



```
package main

import (
    "github.com/FadhlanHawali/Digitalent-Kominfo_Implementation-MVC-
Golang/app/controller"
    "github.com/FadhlanHawali/Digitalent-Kominfo_Implementation-MVC-
Golang/app/middleware"
    "github.com/gin-contrib/cors"
    "github.com/gin-gonic/gin"
)

func main(){

    router := gin.Default()
    router.Use(cors.Default())
    router.POST("/api/v1/account/add", controller.CreateAccount)
    router.POST("/api/v1/login",controller.Login)
    router.GET("/api/v1/account",middleware.Auth,controller.GetAccount)
    router.POST("/api/v1/transfer",middleware.Auth,controller.Transfer)
    router.POST("/api/v1/withdraw",middleware.Auth,controller.Withdraw)
    router.POST("/api/v1/deposit",middleware.Auth,controller.Deposit)
    router.Run(":8080")
}
```