更新说明

by 基础组刘嘉昊

1.router.go

修改import如下，否则不能进行调试

```
import (
    "github.com/gin-gonic/gin"
    "go.mod/controller"
)
```

2. 数据库更新说明

在model/dal.go中的user类中加入一个Password string类型

```
Password string `json:"password,omitempty"`
```

model 文件夹里也加了一个mysql.go，用于数据库的连接等操作

```
//mysql.go
package model

import (
    "github.com/sirupsen/logrus"
    "gorm.io/driver/mysql"
    "gorm.io/gorm"
)

var Mysql *gorm.DB

func init() {
    var err error
    dsn := "root:000729@tcp(127.0.0.1:3306)/douyin?
charset=utf8mb4&parseTime=true&loc=Local"

    logrus.Info("初始化数据库...")

    Mysql, err = gorm.Open(mysql.Open(dsn), &gorm.Config{})
    if err != nil {
        panic("failed to connect database")
    }
}
```

3. token相关说明

在抖音文件夹中加一个utils文件夹，里面放jwt.go 用于token生成

```go
//jwt.go
package utils

import (
    "errors"
    "github.com/dgrijalva/jwt-go"
    "github.com/gin-gonic/gin"
    "net/http"
    "time"
)

// 定义过期时间
const TokenExpireDuration = time.Hour * 24

var MySecret = []byte("这是一段生成token的密钥")

// 用来决定JWT中应该存储哪些数据
type MyClaims struct {
    UserId   int64  `json:"userId"`
    Username string `json:"username"`
    jwt.StandardClaims
}

//生成token并返回

func GenToken(userId int64, username string) (string, error) {
    c := MyClaims{
        userId,
        username,
        jwt.StandardClaims{
            ExpiresAt: time.Now().Add(TokenExpireDuration).Unix(),
            Issuer:    "userFunction",
        },
    }
    token := jwt.NewWithClaims(jwt.SigningMethodHS256, c)
    return token.SignedString(MySecret)
}

//解析JWT
func ParseToken(tokenString string) (*MyClaims, error) {
    token, err := jwt.ParseWithClaims(tokenString, &MyClaims{}, func(token
*jwt.Token) (interface{}, error) {
        return MySecret, nil
    })
    if err != nil {
        return nil, err
    }

    if claims, ok := token.Claims.(*MyClaims); ok && token.Valid {
        return claims, err
    }
    //token失效
    return nil, errors.New("invalid token")
}

// 后续会携带着token进行请求接口

func JWTAuthMiddleware() func(c *gin.Context) {
```

```
    return func(c *gin.Context) {

        token, ok := c.GetQuery("token")

        if !ok {
            c.AbortWithStatusJSON(http.StatusOK, gin.H{
                "status_code": 1,
                "status_msg":  "未携带token",
            })
            return
        }

        res, err := ParseToken(token)

        if err != nil {
            c.JSON(http.StatusOK, gin.H{
                "status_code": 1,
                "status_msg":  err.Error(),
            })
            return
        }
        //保存当前请求信息到上下文c中

        c.Set("user_id", res.UserId)
        c.Set("username", res.Username)

        //继续执行后续的请求
        c.Next()

    }
}
```

4.登录操作相关

在user.go中，把import改成这也（都是泪）

```
import (
    "fmt"
    "github.com/gin-gonic/gin"
    "go.mod/model"
    "go.mod/utils"
    "net/http"
    "sync/atomic"
)
```

加密部分

在controller文件夹中，放一个en_decode.go

```
package controller

import (
    "encoding/base64"
)
```

```
//base64加密方式
func base64Encode(src []byte) []byte {
    return []byte(base64.StdEncoding.EncodeToString(src))
}

//base64解密方式
func base64Decode(src []byte) ([]byte, error) {
    return base64.StdEncoding.DecodeString(string(src))
}
```

在user.go中，接收到传入的密码后，我们对它加密，然后与数据库中储存的已经经过加密的密码进行比较，这也本地储存的就不是真正的密码，减小了风险。

【当然，这要求你注册时就要存储的就是加密后的密码，可以类比Login函数来使用】

如果你想知道如何使用加密和解密函数，请换个地方运行下面的代码

```
package main

import (
    "encoding/base64"
    "fmt"
)

func base64Encode(src []byte) []byte {
    return []byte(base64.StdEncoding.EncodeToString(src))
}

func base64Decode(src []byte) ([]byte, error) {
    return base64.StdEncoding.DecodeString(string(src))
}

func main() {
    // encode
    hello := "123456"
    debyte := base64Encode([]byte(hello))
    fmt.Println(string(debyte))
    // decode
    enbyte, err := base64Decode(debyte)
    if err != nil {
        fmt.Println(err.Error())
    }

    if hello != string(enbyte) {
        fmt.Println("hello is not equal to enbyte")
    }

    fmt.Println(string(enbyte))
}
```