# User控制器文件，需要更新的代码：

## Create：

|  |
| --- |
| //特定数据设置  //生成 密码salt  saltChars := "zxcasdqweghjn@\*&^%#1234567POLK." //salt随机串  saltLen := 6 //salt长度  salt := ""  for i := 0; i < saltLen; i++ {  // 随机数  index := rand.Int31n(int32(len(saltChars)))  // log.Println(index)  salt += string(saltChars[index])  }  //将salt摘要记录  m.PasswordSalt = salt  log.Printf("Salt摘要：%s\n", salt)  //为密码做信息摘要（获取加密器）  pwdFunc := hmac.New(sha256.New, []byte(salt))  //加密数据  pwdFunc.Write([]byte(m.Password)) //向原密码写入摘要密串  m.Password = fmt.Sprintf("%x", pwdFunc.Sum(nil)) //计算生成摘要值  log.Printf("加密后结果：%s\n", m.Password) |

## List：

|  |
| --- |
| //获取product模型  ms := []model.User{}  //获取展示数量和偏移量,输出数据获  orm.Where(condStr, condParams).Order(orderStr).Limit(pageSize).Offset(offset).Find(&ms)  //隐私数据，制空属性，前端不显示密码和密码摘要  for i, \_ := range ms {  ms[i].PasswordSalt = ""  ms[i].Password = ""  } |

## Update：

|  |
| --- |
| // 临时取消关联,并更新数据  orm.Model(&m).  Set("gorm:save\_associations", false).  //仅更新非空字段（前端编辑时无法更改密码和Salt）  Update(m) |

## //用户校验：

|  |
| --- |
| //用户校验  func UserAuth(c \*gin.Context) {  user := model.User{}  //通过用户名获取用户信息，在比较密码是否正确  postUser := c.PostForm("User")  //判断该用户是否不存在，select \* table from user where user=?;  // v := orm.Where("user=?", postUser).Find(&user).RecordNotFound()  // fmt.Println(postUser)  if orm.Where("user=?", postUser).Find(&user).RecordNotFound() {  //没有该用户  c.JSON(http.StatusOK, gin.H{  "error": "用户或密码错误(用户名)",  })  return  }  //用户存在，检查密码  postPassword := c.PostForm("Password")  //通过数据库中存储的Salt，处理密码  pwdFunc := hmac.New(sha256.New, []byte(user.PasswordSalt))  //将前端传递的密码进行摘要算法加密  pwdFunc.Write([]byte(postPassword))  //判断加密后的前端密码和数据库中的密码是否相同  if user.Password != fmt.Sprintf("%x", pwdFunc.Sum(nil)) {  //密码错误  c.JSON(http.StatusOK, gin.H{  "error": "用户或密码错误(密码)",  })  return  }  //生成Token签名key随机串  mySigningKey := []byte("AllYourBase")  // Token内容  claims := &jwt.StandardClaims{  // 有效期，当前时间戳+有效时间(毫秒)  ExpiresAt: time.Now().Unix() + 7200,  //发行人  Issuer: "Backend",  //对应的用户名  Audience: user.User,  }  // 获取token构建器  tokenBuilder := jwt.NewWithClaims(jwt.SigningMethodHS256, claims)  //签名加密生成token  token, err := tokenBuilder.SignedString(mySigningKey)  if err != nil {  c.JSON(http.StatusOK, gin.H{  "error": "Token生成失败",  })  return  }  // fmt.Printf("%v %v", token, err)  //用户名密码正确，认证通过  user.Password, user.PasswordSalt = "", ""  c.JSON(http.StatusOK, gin.H{  "error": "",  "user": user,  "token": token,  })  } |

|  |
| --- |
| import (  "crypto/hmac"  "crypto/sha256"  "fmt"  "log"  "math/rand"  "model"  "net/http"  "strconv"  "time"  "github.com/dgrijalva/jwt-go"  "github.com/gin-gonic/gin"  ) |