# 中山大学数据科学与计算机学院本科生实验报告

| 课程名称 | 现代操作系统应用开发 | 任课老师 | 郑贵锋 |
| --- | --- | --- | --- |
| 年级 | 2017级 | 专业（方向） | 软件工程 |
| 学号 | 17343141 | 姓名 | 姚东烨 |
| 电话 | 13246859092 | Email | 894816193@qq.com |
| 开始日期 | 2019.12.20 | 完成日期 | 2019.12.22 |

## 一、实验题目

基于Go语言的服务端接口开发

## 二、实现内容

1. 设计服务端接口协议，完善信息流接口

- 参考课堂提到的样例代码，独立完成feed流内容接口的定义和开发

2. 设计服务端接口协议及数据库表设计，支持用户注册登录功能

- 完善App页面右下角的“个人中心”，支持输入用户名+密码完成注册、登录的功能
- 支持用户退出登录

3. 设计服务端接口协议及数据库表设计，支持点赞功能

- 每个feed流中的文章，会显示该文章累计获得的点赞总数
- 用户登录状态下， 每个文章会显示用户是否已经点赞过，如果点赞过，则会显示为红色实心的“心”，没有点赞过，则显示空心的“心”。
- 用户可以对文章进行点赞和取消点赞操作。如果用户未登录，则服务端接口返回未登录，客户端页面跳转用户登录注册界面。如果用户已登录，则可以进行点赞和取消点赞。
- 一个登录用户最多只能点赞同一文章一次，对已点赞的文章再次点赞，则相当于进行取消点赞操作。

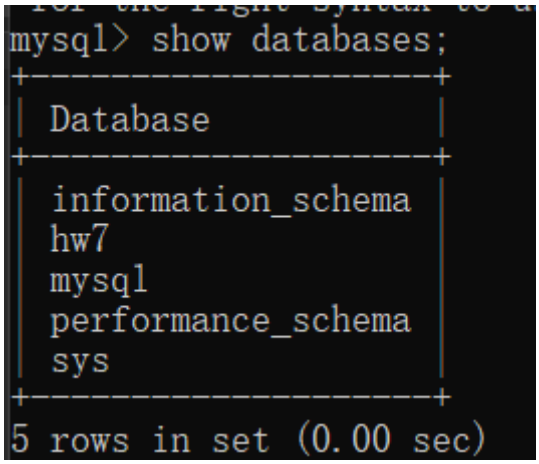
## 三、实验结果

### (1)实验截图

路由设计：

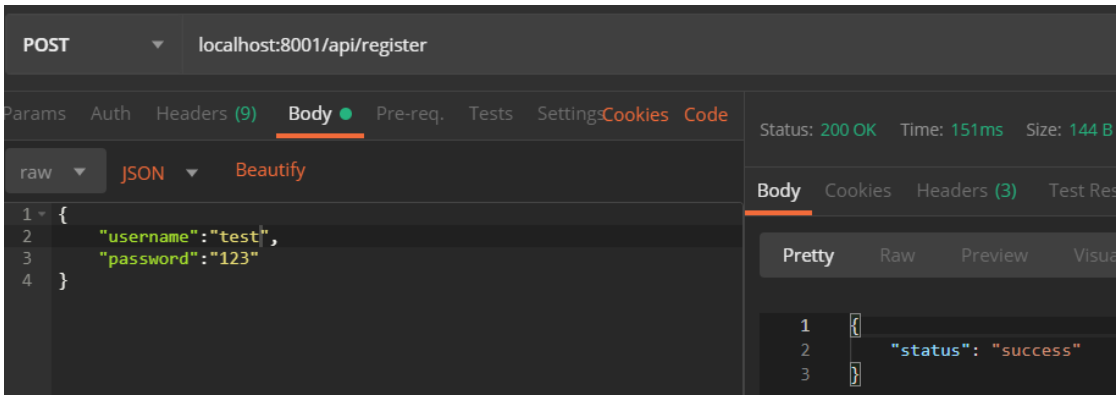
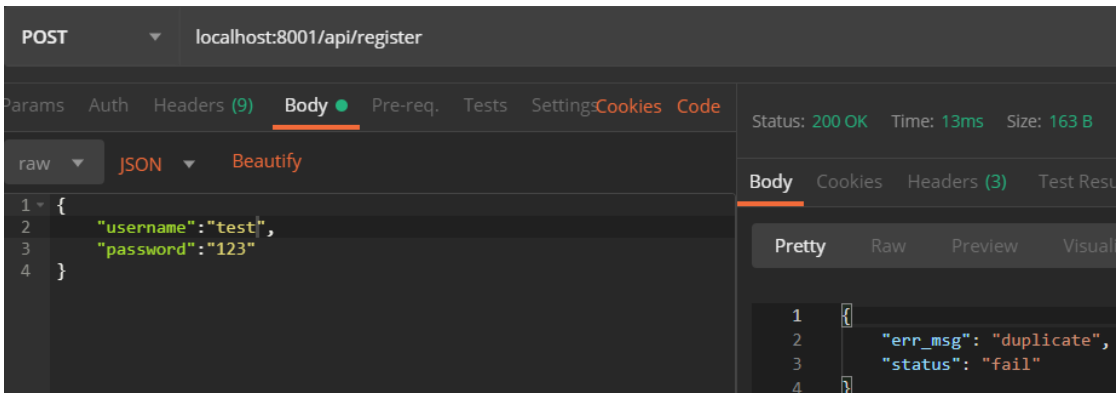
```
[GIN-debug] GET    /api/test                 --> main.Init.func1 (3 handlers)
[GIN-debug] POST   /api/register             --> hw7/api.Register (3 handlers)
[GIN-debug] POST   /api/login                --> hw7/api.Login (3 handlers)
[GIN-debug] POST   /api/logout               --> hw7/api.Logout (3 handlers)
[GIN-debug] GET    /api/type/:tp/user/:username/getFeed --> hw7/api.GetFeed (3 handlers)
[GIN-debug] POST   /api/like                 --> hw7/api.Like (3 handlers)
[GIN-debug] POST   /api/publish              --> hw7/api.Publish (3 handlers)
```

使用的是mysql，并创建数据库hw7：`create database hw7;`



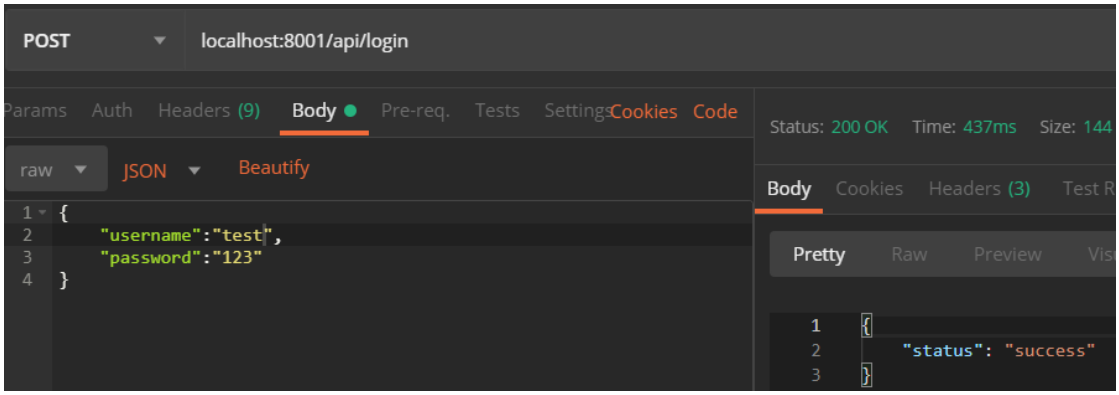

使用Postman进行服务端接口测试：

**用户注册**

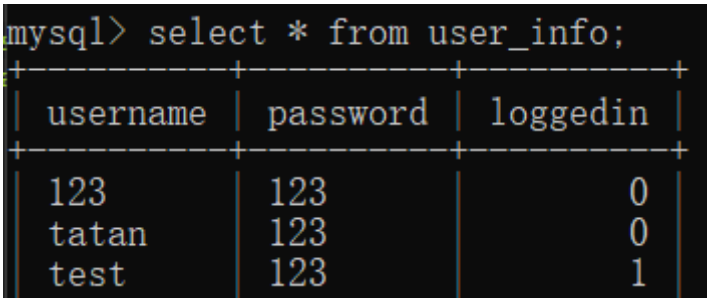
> 首次注册成功：
>
> 
> 
>
> 二次注册失败：
>
> 
> 

**用户登录**

> 登陆成功：



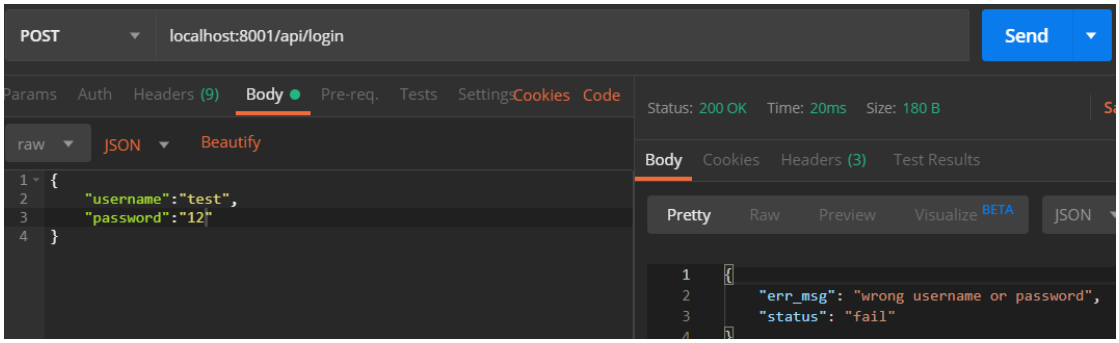

数据库情况：



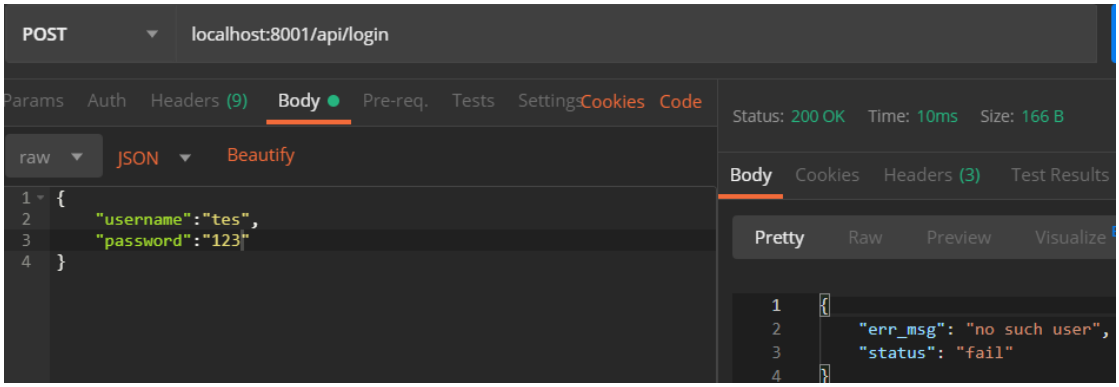
```
mysql> select * from user_info;
+----------+----------+----------+
| username | password | loggedin |
+----------+----------+----------+
| 123      | 123      |        0 |
| tatan    | 123      |        0 |
| test     | 123      |        1 |
+----------+----------+----------+
```

登陆失败：

（用户名/密码错误）

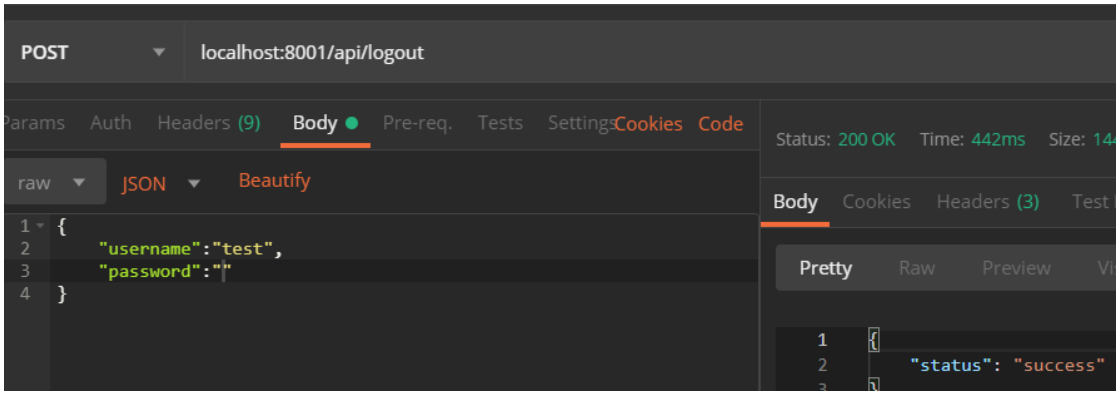



（用户不存在）




**用户登出**

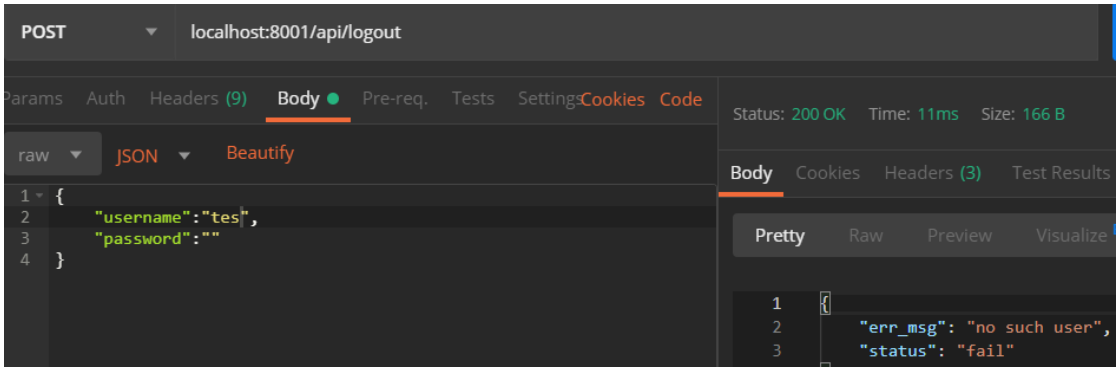
成功：




数据库情况：

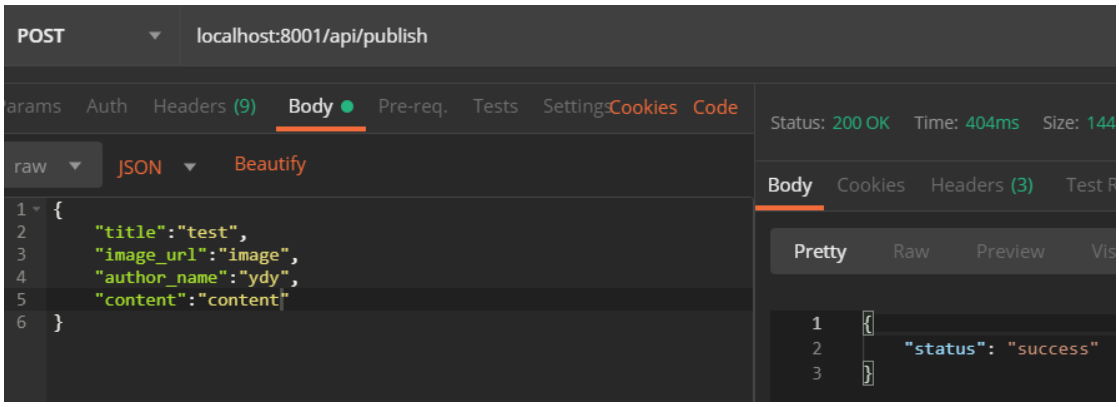
```
mysql> select * from user_info;
+----------+----------+----------+
| username | password | loggedin |
+----------+----------+----------+
| 123      | 123      |        0 |
| tatan    | 123      |        0 |
| test     | 123      |        0 |
```

失败：




**发布文章**

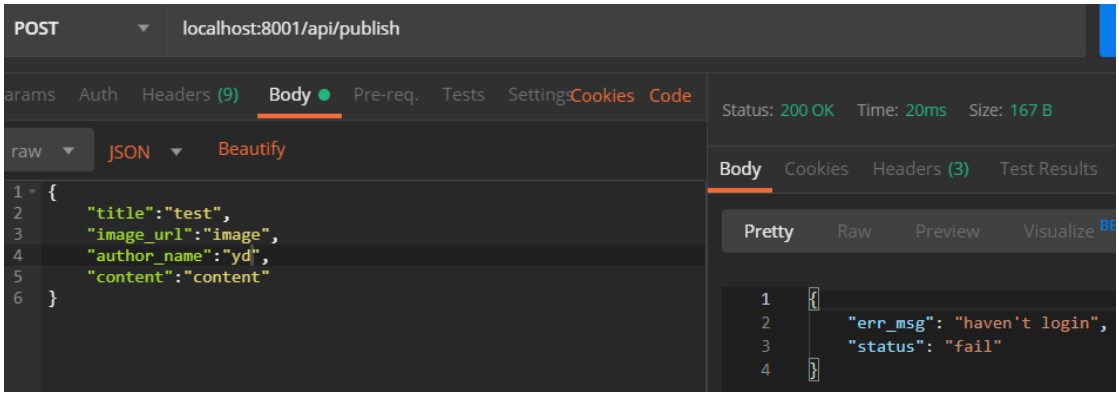
成功：




数据库情况：

```
mysql> select * from article;
+----+-------+-----------+-------------+---------+--------------+
| id | title | image_url | author_name | content | type1_score  |
+----+-------+-----------+-------------+---------+--------------+
|  1 | 1     | 2         | ydy         | 4       |            0 |
|  2 | 1     | 2         | ydy         | 4       |            0 |
|  3 | 1     | 2         | ydy         | 4       |            0 |
|  4 | 1     | 2         | ydy         | 4       |            0 |
|  5 | 1     | 2         | ydy         | 4       |            0 |
|  6 | 1     | 2         | ydy         | 4       |            0 |
|  7 | 1     | 2         | ydy         | 4       |            0 |
|  8 | test  | image     | ydy         | content |            0 |
+----+-------+-----------+-------------+---------+--------------+
```

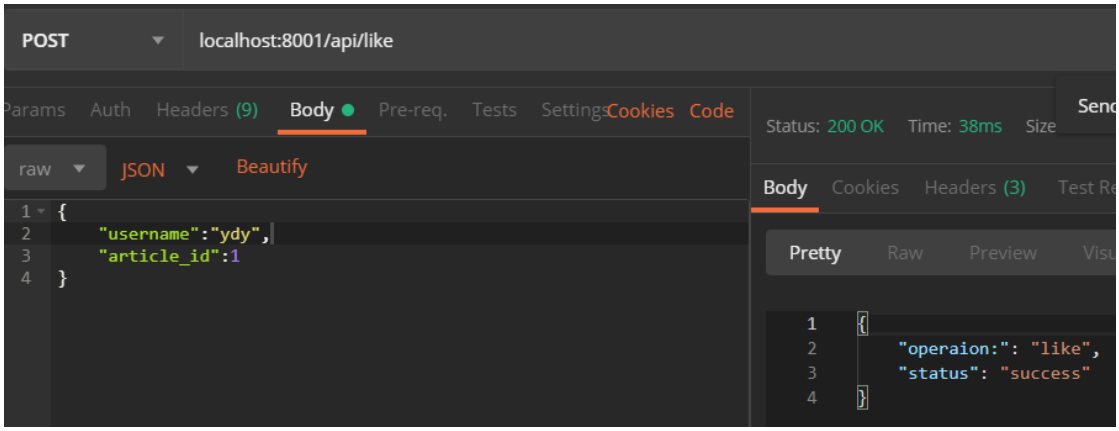
失败:



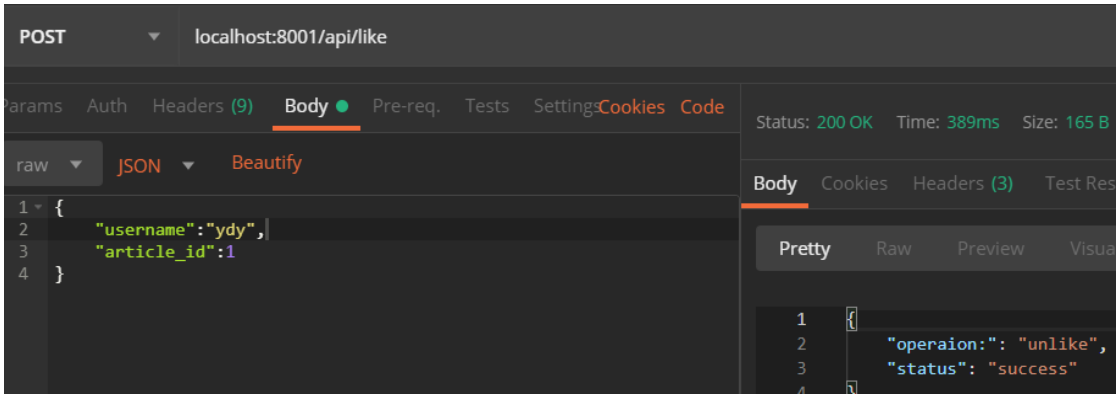

**点赞/取消赞功能**

成功:

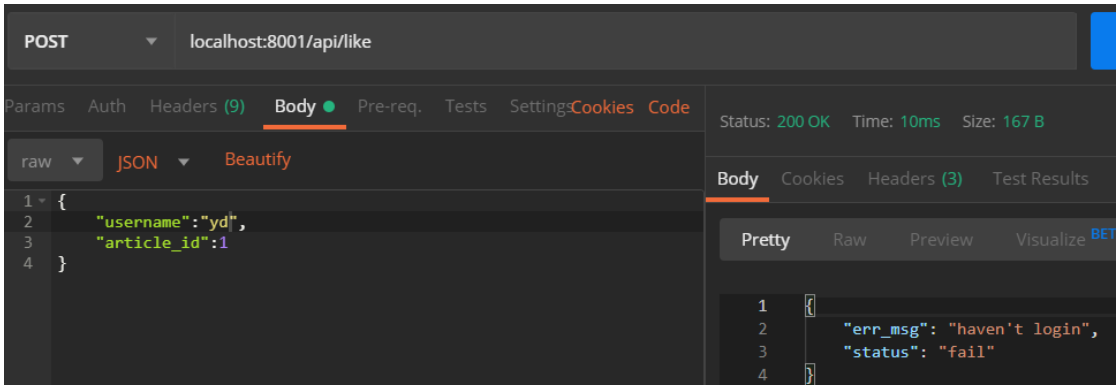
- 点赞:




- 取消赞:

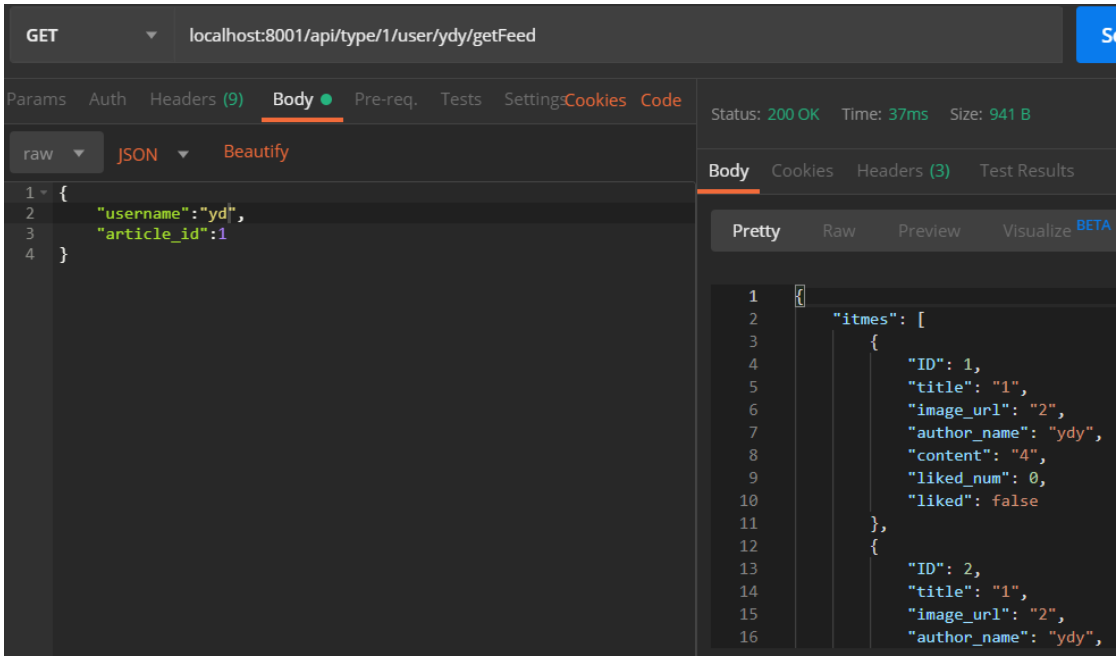



失败:

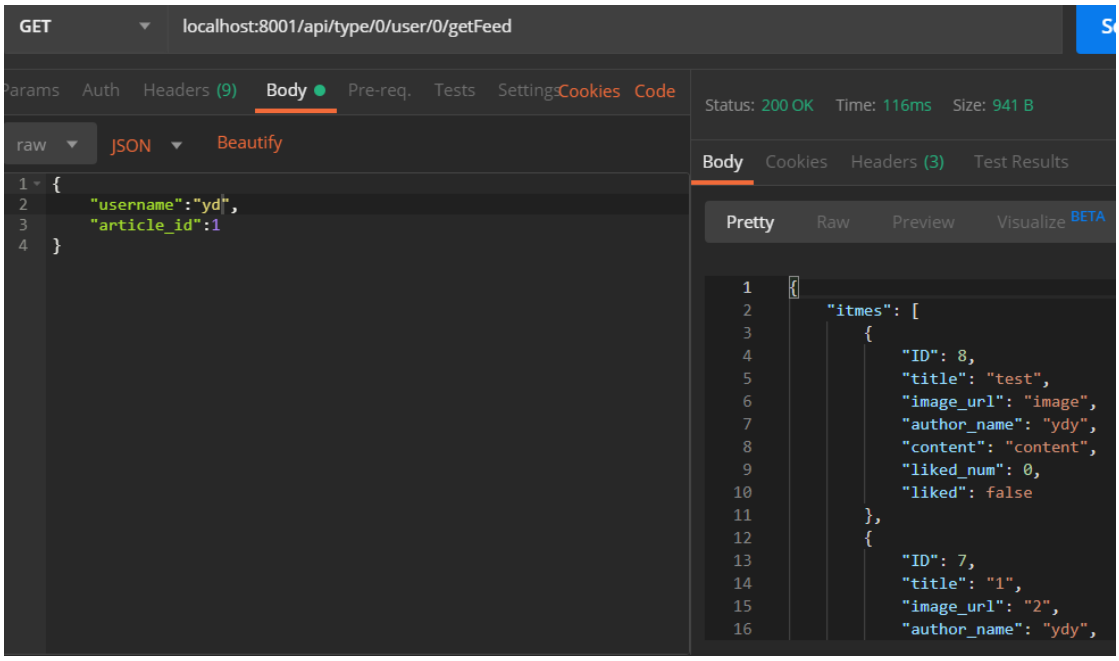



**获得feed流**

已登陆用户：




未登录用户：




# (2)实验步骤以及关键代码

**用户注册功能设计**

查询 **user_info** 表，验证该用户名是否已经使用过了。

查询语句：

```
var duplicate static.UserInfo
db.Where(&receive).First(&duplicate)
```

具体代码如下：

```
func Register(c *gin.Context) {
    var receive static.UserInfo
    c.BindJSON(&receive)

    //get dataBase
    db, err := gorm.Open("mysql", "root:root@tcp(127.0.0.1:3306)/hw7?
charset=utf8")
    db.SingularTable(true)
    if err != nil {
        fmt.Println(err)
    //  panic("Open DB error!")
    }
    defer db.Close()
    db.AutoMigrate(&static.UserInfo{})

    //check whether if duplicate
    var duplicate static.UserInfo
    db.Where(&receive).First(&duplicate)
    if duplicate==(static.UserInfo{}){
        db.Create(&receive)
        c.JSON(http.StatusOK, gin.H{
            "status":  "success",
        })
    }else{
        c.JSON(http.StatusOK, gin.H{
            "status":  "fail",
            "err_msg": "duplicate",
        })
    }
}
```

### 用户登录功能设计

首先查询 **user_info** 表，验证用户是否存在

查询语句：

```
var user static.UserInfo
db.Where("username = ?", receive.Username).First(&user)
```

接着判断用户密码是否匹配，匹配则修改对应条目中的 **loggedin** 值为 **true** 返回成功，否则返回失败。

具体代码如下：

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

```
    var receive static.UserInfo
    c.BindJSON(&receive)
    fmt.Println(receive)
    //get dataBase
    db, err := gorm.Open("mysql", "root:root@tcp(127.0.0.1:3306)/hw7?
charset=utf8")
    db.SingularTable(true)
    if err != nil {
        fmt.Println(err)
    //  panic("Open DB error!")
    }
    defer db.Close()
    db.AutoMigrate(&static.UserInfo{})

    //get the userinfo
    var user static.UserInfo
    db.Where("username = ?", receive.Username).First(&user)
    //check whether the user exist
    if user==(static.UserInfo{}){
        c.JSON(http.StatusOK, gin.H{
            "status":  "fail",
            "err_msg": "no such user",
        })
    }else{
        //check whether the password is correct
        if user.Password != receive.Password{
            c.JSON(http.StatusOK, gin.H{
                "status":  "fail",
                "err_msg": "wrong username or password",
            })
        }else{
            db.Delete(&user)
            user.Loggedin=true
            db.Create(&user)
            c.JSON(http.StatusOK, gin.H{
                "status":  "success",
            })
        }
    }
}
```

**用户登出功能**

查询 **user_info** 表，查看是否存在该用户

查询语句：

```
var user static.UserInfo
db.Where("username = ?", receive.Username).First(&user)
```

若存在则允许登出，修改对应条目的 **loggedin** 为 **false** ，否则返回错误信息

具体代码如下：

```
func Logout(c *gin.Context) {
    var receive static.Logout
```

```
    c.BindJSON(&receive)

    //get dataBase
    db, err := gorm.Open("mysql", "root:root@tcp(127.0.0.1:3306)/hw7?
charset=utf8")
    db.SingularTable(true)
    if err != nil {
        fmt.Println(err)
    //  panic("Open DB error!")
    }
    defer db.Close()
    db.AutoMigrate(&static.UserInfo{})

    //get the userinfo
    var user static.UserInfo
    db.Where("username = ?", receive.Username).First(&user)
    //check whether the user exist
    if user==(static.UserInfo{}){
        c.JSON(http.StatusOK, gin.H{
            "status":  "fail",
            "err_msg": "no such user",
        })
    }else{
        db.Delete(&user)
        user.Loggedin=false
        db.Create(&user)
        c.JSON(http.StatusOK, gin.H{
            "status":  "success",
        })
    }
}
```

**发布文章功能**

首先查询 **user_info** 表，确认用户是否登陆

查询语句如下：

```
var user static.UserInfo
db.Where("username = ? AND loggedin = ?", receive.Author_name,
true).Find(&user)
```

若已登录，则允许发布文章，在 **article** 表中添加记录，否则返回错误信息。

具体代码如下：

```
func Publish(c *gin.Context) {
    var receive static.PublishInfo
    c.BindJSON(&receive)
    //get dataBase
    db, err := gorm.Open("mysql", "root:root@tcp(127.0.0.1:3306)/hw7?
charset=utf8")
    db.SingularTable(true)
    if err != nil {
        fmt.Println(err)
    //  panic("Open DB error!")
```

```
    }
    defer db.Close()
    db.AutoMigrate(&static.Article{})

    var user static.UserInfo
    db.Where("username = ? AND loggedin = ?", receive.Author_name,
true).Find(&user)
    if user == (static.UserInfo{}){
        c.JSON(http.StatusOK, gin.H{
            "status":  "fail",
            "err_msg": "haven't login",
        })
        return
    }
    // make the article to publish
    var article static.Article
    article.Title = receive.Title
    article.Image_url = receive.Image_url
    article.Author_name = receive.Author_name
    article.Content = receive.Content

    db.Create(&article)
    c.JSON(http.StatusOK, gin.H{
        "status":  "success",
    })
}
```

**点赞功能**

首先查询 **user_info** 表，确认是否登陆

查询语句如下：

```
var user static.UserInfo
db.Where("username = ? AND loggedin = ?", receive.Username,
true).Find(&user)
```

若登陆成功，则查询 **Article** 表，确认点赞文章是否存在

查询语句如下：

```
var exist static.Article
db.Where("ID = ?", receive.Article_id).Find(&exist)
```

若不存在则返回错误信息，若存在则查询 **like_info** 表，判断该点赞还是取消赞

查询语句如下：

```
var like static.LikeInfo
db.Where("username = ? AND article_id = ?", receive.Username,
receive.Article_id).Find(&like)
```

若存在这条记录，则应删除该条记录，否则应该生成一条新记录

具体代码如下：

```
func Like(c *gin.Context) {
    var receive static.LikeInfo
    c.BindJSON(&receive)
    fmt.Println(receive)
    //get dataBase
    db, err := gorm.Open("mysql", "root:root@tcp(127.0.0.1:3306)/hw7?
charset=utf8")
    db.SingularTable(true)
    if err != nil {
        fmt.Println(err)
    //  panic("Open DB error!")
    }
    defer db.Close()
    db.AutoMigrate(&static.LikeInfo{})

    //check whether the user is OK
    var user static.UserInfo
    db.Where("username = ? AND loggedin = ?", receive.Username,
true).Find(&user)
    if user == (static.UserInfo{}){
        c.JSON(http.StatusOK, gin.H{
            "status":  "fail",
            "err_msg": "haven't login",
        })
        return
    }


    //check whether if the article exists
    var exist static.Article
    db.Where("ID = ?", receive.Article_id).Find(&exist)
    if exist == (static.Article{}){
        c.JSON(http.StatusOK, gin.H{
            "status":  "fail",
            "err_msg": "article has been deleted",
        })
    }else{
        var like static.LikeInfo
        db.Where("username = ? AND article_id = ?", receive.Username,
receive.Article_id).Find(&like)
        if like == (static.LikeInfo{}){
            db.Create(&receive)
            c.JSON(http.StatusOK, gin.H{
                "status":  "success",
                "operaion:": "like",
            })
        }else{
            db.Delete(&like)
            c.JSON(http.StatusOK, gin.H{
                "status":  "success",
                "operaion:": "unlike",
            })
        }
    }
}
```

**获得feed流**

> 首先根据传过来的 **type** 以及 **username** 确定是否为登陆用户。
>
> 若是未登录用户，使用 **article** 表的 **ID** 进行降序排序选取最多前10条返回，否则按照用户喜欢的 **type** 进行降序排序，再选取最多前10条返回
>
> 查询语句如下：

```
var articles []static.Article
    if tp=="1"{
        db.Select("*").Order("type1_score desc").Limit(10).Find(&articles)
    } else if tp=="0"{
        db.Select("*").Order("ID desc").Limit(10).Find(&articles)
    }
```

> 具体代码如下：

```
func GetFeed(c *gin.Context) {
    tp := c.Param("tp")
    name := c.Param("username")
    //get dataBase
    db, err := gorm.Open("mysql", "root:root@tcp(127.0.0.1:3306)/hw7?
charset=utf8")
    db.SingularTable(true)
    if err != nil {
        fmt.Println(err)
    //  panic("Open DB error!")
    }
    defer db.Close()
    db.AutoMigrate(&static.Article{})

    //find the top 10 articles as desc with the corresponding type
    var articles []static.Article
    if tp=="1"{
        db.Select("*").Order("type1_score desc").Limit(10).Find(&articles)
    } else if tp=="0"{
        db.Select("*").Order("ID desc").Limit(10).Find(&articles)
    }
    feedList := make([]static.ArticleForShow, 0)
    var likeinfo static.LikeInfo
    for i:= 0; i<len(articles); i++{
        var tmp static.ArticleForShow
        tmp.ID=articles[i].ID
        tmp.Title=articles[i].Title
        tmp.Image_url=articles[i].Image_url
        tmp.Author_name=articles[i].Author_name
        tmp.Content=articles[i].Content
        //get the like num
        db.Where("article_id = ?",
tmp.ID).Find(&likeinfo).Count(&tmp.Liked_num)
        var tmptmp static.LikeInfo
        //check whether if the current user like
        db.Where("article_id = ? AND username = ?", tmp.ID,
name).Find(&tmptmp)
        if tmptmp == (static.LikeInfo{}){
            tmp.Liked = false
```

```
        }else{
            tmp.Liked = true
        }
        feedList=append(feedList,tmp)
     }
    c.JSON(http.StatusOK, gin.H{
        "itmes":feedList,
    })
}
```

**路由设置**

```
be_api := router.Group("/api")
{
    be_api.GET("/test", func(c *gin.Context) {
        fmt.Println("test...")
        c.JSON(200, gin.H{
            "msg": "OK!",
        })
    })
    be_api.POST("/register", api.Register)
    be_api.POST("/login", api.Login)
    be_api.POST("/logout", api.Logout)
    be_api.GET("/type/:tp/user/:username/getFeed",api.GetFeed)
    be_api.POST("/like",api.Like)
    be_api.POST("/publish",api.Publish)
}
```

其中 `/api/test` 是供测试的时候排查网络问题所用的。

## (3)实验遇到的困难以及解决思路

- **gorm包拉不下来**

  在我跟着教程学习完gorm之后，准备go get gorm包来进行实验时，发现我们的校园网总是报超时，所以准备偷懒，使用boltdb来进行实验。但是当我询问了老师的意见后，老师认为我们应该使用远程数据库来完成实验，所以根本原因还是这个gorm一定要拉下来。接着我尝试了使用虚拟机，并使用手机热点来进行拉去包的操作，试了好几次之后才将包拉下来，这样才开始了愉快的编程工作。

- **数据库中为0的值会被忽略**

  当我进行编程的时候，发现我想将 **loggedin** 从 **false** 改成 **true** 的时候总是改不了，后面通过网络查询才发现update的时候为0的值会被忽略，所以我选择了将这条记录删除，并重新插入一个更改值后的记录。这样解决了问题。

- **map无法存入json**

  在我考虑如何将点赞信息存储时，一开始我是想着将点赞信息存在文章的结构体中，但是运行的时候会报 map 无法直接被存进 json 的问题。我首先想到的是将 map 转化为 json，接着 json 作为一个 string 存进去，之后要取出来的时候再做操作。但是后来我觉得这样做太麻烦了，所以还是选择建立一个记录点赞信息的表。这样解决了问题。

# 四、实验思考及感想

本次实验是我第一次独立完成一个基本的服务端设计，我觉得整个过程非常有趣，特别是在我将所有的接口做好，通过Postman来测试我的接口时，那种成就感是十分强烈的。在这次实验中，我学会了gin这个非常简单易操作的框架，让我能非常简单直观的处理前端发来的请求。同时，我也基本地掌握了gorm的简单操作。一开始看gorm的教程的时候感觉十分混乱，但是只要静下心来，打开vscode边看边试，整个学习就可以做下来了，其实入门也不是很难，高级的操作我们本次实验中还不需要使用。

比较遗憾的是，本次实验中我没有完成加分项，即用户评论的功能。因为本次实验是与课程设计并行的，而课程设计是一个团队项目，需要尽量的配合其他组员的时间安排，所以就只能选择割舍这个hw7的一部分时间，导致最后脑子有点炸裂，没时间完成用户评论的功能。其实用户评论我的基本构思也是想着和like一样，存在另一个表中，然后通过ID来确定评论的顺序。但是回复评论这个功能我还是没有构思好，所以也没敢直接动手开始做，怕到时候搞得一坨浆糊更不好改了。临近期末，各种大作业也蜂拥而至，等到假期到的时候一定要将这个评论功能完成，并连接前端真正做成一个小app。

总而言之，本次实验教会了我许多使用的框架、工具，我会把他们用到我之后的工作学习中，希望在之后的工作学习中能做的更好吧。