

抖音项目汇报文档

项目名称	抖音项目[青训营]
当前版本	版本 V1.0
开发语言	Golang
完成时间	2022.06.11
开发团队	八方旅人队
队号	970521
团队成员	杨杰、孙小涵、曾韩韬 江小白、张成明 杨雨孜、阎鸿彬、吕丹瑜

目录

- 抖音项目汇报文档1
 - 一 引言3
 - 1.1 项目背景3
 - 1.2 项目完成情况3
 - 1.3 项目成员分工4
 - 二 项目灵感 亮点5
 - 2.1 项目灵感5
 - 2.2 项目亮点5
 - 三 项目技术说明6
 - 3.1 使用 Gorm 操作数据库6
 - 3.2 使用 Gin 框架处理请求6
 - 3.3 使用静态文件存储数据8
 - 3.4 使用 ffmpeg 生成视频封面9
 - 3.5 缓存处理 Redis9
 - 四 难点突破11
 - 4.1 用户身份验证令牌 Token11
 - 4.2 点赞操作12
 - 4.3 代码测试12
 - 五 成果展示15
 - 5.1 代码展示15
 - 5.2 项目运行18
 - 六 项目总结22
 - 6.1 总结22
 - 6.2 展望22

一 引言

1.1 项目背景

实现极简版抖音，通过使用 Go 语言编程，使用 gin 框架，结合数据库和对象存储等内容，实现相对完善的抖音后端。使得 APK 客户端能够调用相关接口实现用户使用抖音基本功能，如刷视频、注册登录、点赞评论等等。

实践在青训营期间的课程知识，并提高对开发工作的了解和认识，促进个人技术成长以及启发视野。

1.2 项目完成情况

功能项	说明	实现
视频 Feed 流、投稿、个人信息	支持所有用户刷抖音，登录用户可以投稿，查看自己信息	已实现
点赞列表、用户评论	登录用户能点赞、评论，个人页能看点赞视频列表	已实现
关注列表、粉丝列表	登录用户能关注其他用户，能点击关注、粉丝列表	已实现

1.3 项目成员分工

1. 视频 Feed 流、投稿、个人信息（基础功能）

杨杰、孙小涵、曾韩韬

2. 点赞功能、用户评论（拓展功能 I）

江小白、张成明

3. 关注列表、粉丝列表（拓展功能 II）

杨雨孜、阎鸿彬、吕丹瑜

二 项目灵感 亮点

2.1 项目灵感

抖音 APP 作为当下最火的短视频应用，有着广大的使用群体和强大的功能，研究抖音项目的开发过程能够让我们学习了解商业产品的开发流程，并且将学习的知识进行实践。

对我们团队成员来说，能够将青训营这段时间所学应用到实际开发，不仅能够提升我们动手编程开发的能力，夯实基础知识并不断发现问题解决问题，而且能够提升我们的团队协作能力和解决问题的能力，为我们以后的工作打下良好基础。而且能够进一步了解抖音项目背后的知识也极大的吸引了我们求知的动力，因此我们团队选择抖音项目的开发。

2.2 项目亮点

本项目的亮点，首先完成要求的所有功能，能够实现顺利刷视频、用户注册登录、上传视频、点赞评论和关注，保证用户能有完整的使用体验。其次使用 Gorm 让我们开发过程变得简便快捷，能够方便操作数据库。通过使用 Token 能极大保护用户身份信息且检测快速。最重要的是在多个客户端同时使用时，也能保障后端性能和操作正确性，提升了用户满意度。

三 项目技术说明

3.1 使用 Gorm 操作数据库

ORM (Object-Relational Mapping) 对象关系映射模式是一种为了解决面向对象与关系数据库存在的互不匹配的现象的技术, 通过使用描述对象和数据库之间映射的元数据, 将程序中的对象自动持久化到关系数据库中。

GORM 是 github 上活跃度很高, 使用广泛的 ORM 库, 通过 GORM 能够让我们在具体操作数据库的时候不用写复杂的 SQL 语句, 能够方便快捷的完成数据库操作。

在编写结构体时, 能够使用 gorm 标签来进行标识索引、外键等信息, 从而在创建对应的数据表时尤为方便。

```
type Video struct {  
    gorm.Model  
    AuthorID int64 `gorm:"not null; index:idx_author_id" json:"author_id"`  
    Title     string  `gorm:"not null" json:"title"`  
    PlayURL   string  `gorm:"not null" json:"play_url"`  
    CoverURL  string  `gorm:"not null" json:"coverurl"`  
}
```

You, 3周前 • 新增项目结构, 新增创建数据库函数

图 1 通过 GORM 创建索引等信息

3.2 使用 Gin 框架处理请求

Gin 是用 Go 编写的 HTTP Web 框架, 运行速度快, 有分组的路由器, 良好的 panic recover 和错误处理, 非常好的支持中间件和数据绑定处理。

在本项目中, 使用 Gin 框架结合 RESTful API 接口, 能够接收客

户端发送的相应请求并使用对应的函数进行处理，根据处理结果的成功与否，返回对应的响应数据或错误信息。

为了减少冗余的 token 处理，我们按照接口是否需要 token 鉴权以及 token 的传入位置，将接口组织为三个组：无需授权组、Query 参数组、form 组，通过认证中间件处理。

对需要鉴权的接口，我们将其按照 token 的传递方式分别组织成两个 group，使用对应的认证 middleware，大大减少了冗余的 token 处理，同时也提高了代码的可维护性。认证 middleware 会中断鉴权失败的错误返回错误码，当鉴权完成后，我们将用户的 id 写入到 gin 的 context 中，在编写 controller 的处理逻辑时候只需要通过 get 方法即可得到传递的 token claims 的 ID。

```
// Unauthorized APIs
noAuthAPIs := r.Group("/douyin")
{
    // public directory is used to serve static resources
    r.Static("/static", "./public")

    // user module
    noAuthAPIs.GET("/feed/", controller.Feed)
    noAuthAPIs.POST("/user/register/", controller.Register)
    noAuthAPIs.POST("/user/login/", controller.Login)
}
```

图 2 无需鉴权的接口

```
// Form auth APIs
formAuthAPIs := r.Group("/douyin")
formAuthAPIs.Use(controller.FormAuthMiddleware)
{
    formAuthAPIs.POST("/publish/action/", controller.Publish)
}
```

图 3 form 表单传递 token 的接口

```
// Query auth APIs
queryAuthAPIs := r.Group("/douyin")
queryAuthAPIs.Use(controller.QueryAuthMiddleware)
{
    // user module
    queryAuthAPIs.GET("/user/", controller.UserInfo)
    // publish module
    queryAuthAPIs.GET("/publish/list/", controller.PublishList)
    // favorite module
    queryAuthAPIs.POST("/favorite/action/", controller.FavoriteAction)
    queryAuthAPIs.GET("/favorite/list/", controller.FavoriteList)
    // comment module
    queryAuthAPIs.POST("/comment/action/", controller.CommentAction)
    queryAuthAPIs.GET("/comment/list/", controller.CommentList)
    // relation module
    queryAuthAPIs.POST("/relation/action/", controller.RelationAction)
    queryAuthAPIs.GET("/relation/follow/list/", controller.FollowList)
    queryAuthAPIs.GET("/relation/follower/list/", controller.FollowerList)
}
```

图 4 Query 参数传递表单的接口

3.3 使用静态文件存储数据

Gin 框架提供静态文件服务,通过指定某个目录为静态资源目录,便能使用 URL 够直接访问这个目录中具体资源。

在项目实践当中,指定 public 目录为静态资源目录,在使用中用户上传的视频文件将存储在 public/video 目录下;此外根据视频生成的封面图片将存储在 public/cover 目录下;并将对应的路径添加好前后缀放入到数据库之中。

当客户端使用 URL 进行访问,便可以返回对应视频和图片信息到客户端进行解析。

3.4 使用 ffmpeg 生成视频封面

FFmpeg 是一个开放源代码的自由软件，可以执行音频和视频多种格式的录影、转换、串流功能。

在 GitHub 上找到对应的 Go 版本仓库，并且下载 ffmpeg.exe 用于截取视频帧作为封面，然后通过官方的样例进行修改得到如下函数。

```
func GenerateCover(videoPath, coverPath string, frameNum int) (string, error) {  
    coverName := ""  
    buf := bytes.NewBuffer(nil)  
    err := ffmpeg.Input(videoPath).  
        Filter("select", ffmpeg.Args{fmt.Sprintf("gte(n,%d)", frameNum)}).  
        Output("pipe:", ffmpeg.KwArgs{"vframes": 1, "format": "image2", "vcodec": "mjpeg"}).  
        WithoutOutput(buf, os.Stdout).  
        Run()  
}
```

图 5 生成封面

VideoPath 表示视频存放的路径，coverPath 表示生成封面需要存储的路径，frameNum 表示需要截取的帧数，一般选择第一帧即可。之后通过设定对应的参数调用 ffmpeg 进行生成封面。

3.5 缓存处理 Redis

Redis 是一个数据库，不过与传统数据库不同的是 Redis 的数据库是存在内存中，所以读写速度非常快，因此 Redis 被广泛应用于缓存方向。将用户访问的数据存在缓存中，这样下一次再访问这些数据的时候就可以直接从缓存中获取了。操作缓存就是直接操作内存，所以速度相当快。并且直接操作缓存能够承受的请求是远远大于直接访问数据库的，可以将数据库部分数据转移到缓存中，使得用户部分请求不经过数据库。

扩展接口 I 中评论操作，可以增删一条评论，而查看视频评论列表的需求则是直接返回一个视频的所有评论，并没有分页处理，所以可以认为对评论的操作是读多写少的。因此，为了提高查询速度，我们可以对评论进行缓存，采用 Redis 缓存评论信息，减少对关系型数据库 MySQL 的查询，从而提高吞吐量。

四 难点突破

4.1 用户身份验证令牌 Token

Token 是服务端生成的一串加密字符串、以作客户端进行请求的一个“令牌”。当用户第一次使用账号密码成功进行登录后，服务器便生成一个 Token 及 Token 失效时间并将此返回给客户端，若成功登陆，以后客户端只需在有效时间内带上这个 Token 前来请求数据即可，无需再次带上用户名和密码。

生成 Token 的难点在于，需要设置有效时间和用户身份信息，并且通过加密函数对其数据进行保护。此外对一个给定的 Token 需要能够进行正确的解析，得到对应的数据。

在项目中具体实现中，使用了 golang-jwt 库，能够在生成 Token 时对信息进行加密，并且使用私钥进行签名保障 Token 的正确合法性。

```
// signJWT signs a JWT and returns it.
func signJWT(userID int64) (string, error) {
    expireTime := time.Now().Add(time.Duration(config.C.JWT.ExpireMinutes) * time.Minute)
    claims := Claims{
        StandardClaims: jwt.StandardClaims{
            Id:        fmt.Sprintf("%d", userID),
            ExpiresAt: expireTime.Unix(),
        },
    },
}
token := jwt.NewWithClaims(jwt.SigningMethodHS256, &claims)
return token.SignedString(secretKey)
}

// parseToken verify a JWT string and returns its claims.
func parseToken(tokenString string) *Claims {
    var claims Claims
    token, err := jwt.ParseWithClaims(tokenString, &claims, func(t *jwt.Token) (interface{}, error) {
        return secretKey, nil
    })
    if err != nil || !token.Valid {
        return nil
    }
    return token.Claims.(*Claims)
}
```

图 6 处理 jwt 签发和验证

4.2 点赞操作

点赞操作的难点主要在于判断点赞和取消点赞的合法性，比如只能点赞视频表中存在的视频，并且该视频之前没有被点赞过，同样也只能取消点赞视频表中存在的视频，并且该视频之前没有被取消点赞。

这里通过设置外键来实现，如下图所示，点赞表 Favorite 中通过 gorm.foreignkey 设置外键，关联视频表 and 用户表之后就能够保证增加点赞记录时 user_id, video_id 必须在 User 表、Video 表中出现过，删除点赞同理。

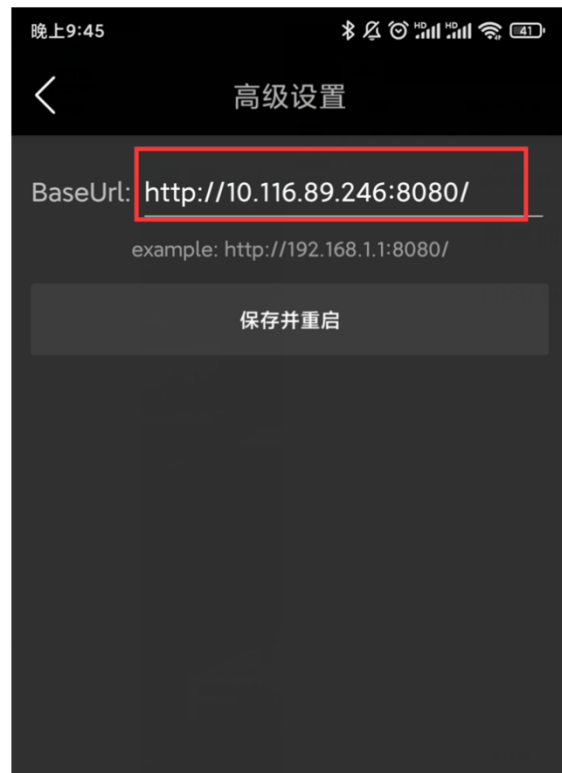
```
// 用户点赞表
// idx_userid_video_id: 查找用户点赞列表, 查找用户是否给某视频点了赞
type Favorite struct {
    gorm.Model
    UserID int64 `gorm:"not null; index:idx_userid_video_id" json:"user_id"`
    VideoID int64 `gorm:"not null; index:idx_userid_video_id" json:"video_id"`
    Video Video `gorm:"foreignkey:VideoID;association_foreignkey:ID"`
    User User `gorm:"foreignkey:UserID;association_foreignkey:ID"`
}
```

4.3 代码测试

编写代码之后测试是验证代码逻辑正确的重要环节，因此如何让代码运行展示效果是一个重要难点。

为了让项目使用起来和抖音有类似的体验，这里首先在手机上安装好 APP，然后对代码项目进行编译运行。接着为了让项目能够监听到 APP 的请求信息，需要打开电脑的热点，并让手机连接上对应热点。其次在网络设置当中打开手动代理，端口号设置为 8080。最后需要在抖声的设置当中修改配置，使其访问的服务器地址为电脑 ip 以及

8080 端口。



[GIN-debug]	GET	/douyin/comment/list/ --> github.com/YJ9938/DouYin/controller.CommentList (3 handlers)				
[GIN-debug]	POST	/douyin/relation/action/ --> github.com/YJ9938/DouYin/controller.RelationAction (3 handlers)				
[GIN-debug]	GET	/douyin/relation/follow/list/ --> github.com/YJ9938/DouYin/controller.FollowerList (3 handlers)				
[GIN-debug]	GET	/douyin/relation/follower/list/ --> github.com/YJ9938/DouYin/controller.FollowerList (3 handlers)				
[GIN-debug] [WARNING]	You trusted all proxies, this is NOT safe. We recommend you to set a value.					
Please check https://pkg.go.dev/github.com/gin-gonic/gin#readme-don-t-trust-all-proxies for details.						
[GIN-debug]	Environment variable PORT is undefined. Using port :8080 by default					
[GIN-debug]	Listening and serving HTTP on *:8080					
[GIN]	2022/06/11 - 11:11:53	200	50.5336ms	10.116.89.246	GET	"/static/video/share_de0ff129671720846fc97e80bdb4854d.mp4"
[GIN]	2022/06/11 - 11:11:53	200	69.172ms	10.116.89.246	GET	"/douyin/favorite/list/?token=eYjHbcGciO1JIuz1NiIsInR5cCI6IjEwMk&userid=15"
TISImpoasIGijuiFq.eNessfF2tCjT0WbkEvhtZjnIByeQyK7s4zB4EmJ0MK&userid=15"						
[GIN]	2022/06/11 - 11:11:54	200	922.7846ms	10.116.89.246	GET	"/static/video/share_de0ff129671720846fc97e80bdb4854d.mp4"
[GIN]	2022/06/11 - 11:11:55	200	0s	10.116.89.246	GET	"/douyin/publish/list/?token=eYjHbcGciO1JIuz1NiIsInR5cCI6IjEwMk&userid=14"
TISImpoasIGijuiFq.eNessfF2tCjT0WbkEvhtZjnIByeQyK7s4zB4EmJ0MK&userid=14"						
[GIN]	2022/06/11 - 11:11:55	200	25.882ms	10.116.89.246	GET	"/douyin/favorite/list/?token=eYjHbcGciO1JIuz1NiIsInR5cCI6IjEwMk&userid=14"
TISImpoasIGijuiFq.eNessfF2tCjT0WbkEvhtZjnIByeQyK7s4zB4EmJ0MK&userid=14"						
[GIN]	2022/06/11 - 11:11:55	200	46.4955ms	10.116.89.246	GET	"/static/video/share_8bdcc2fa85de7885a7da325f586d4becf.mp4"



从上图可看出，成功运行后开始监听访问 8080 端口的请求，能正确处理用户提交的请求，能正常播放视频信息，展示点赞和评论等信息。

五 成果展示

经过小组成员的分工合作，在经历了一个月左右的时间，我们尽可能的完善了代码，并且能够在手机上正常使用，符合项目要求的对应功能。下表展示开发项目的相关信息。

运行环境	代码框架	开发工具 语言	库文件	数据库
Redmi K20Pro 骁龙 855	Gin 框架	VSCode	Gorm	MySQL
MIUI 12.5.6 Android 11		Golang	Jwt	

5.1 代码展示

这一部分将会展示重要的代码块，并做简单的描述，从而更好展示整个项目。

1.Main 文件，首先使用 Gin 的默认函数创建默认引擎，然后进行数据库初始化，并设置好接收路由的操作，最后开始监听 8080 端口。

```
package main

import (
    "github.com/YJ9938/DouYin/model"
    "github.com/YJ9938/DouYin/router"
    "github.com/gin-gonic/gin"
)

func main() {
    r := gin.Default()

    model.InitMySQL()

    router.InitRouter(r)

    r.Run() // listen and serve on 0.0.0.0:8080 (for windows "localhost:8080")
}
```

2.Router 文件，创建 douyin 路由组，并开始监听对应的 GET、

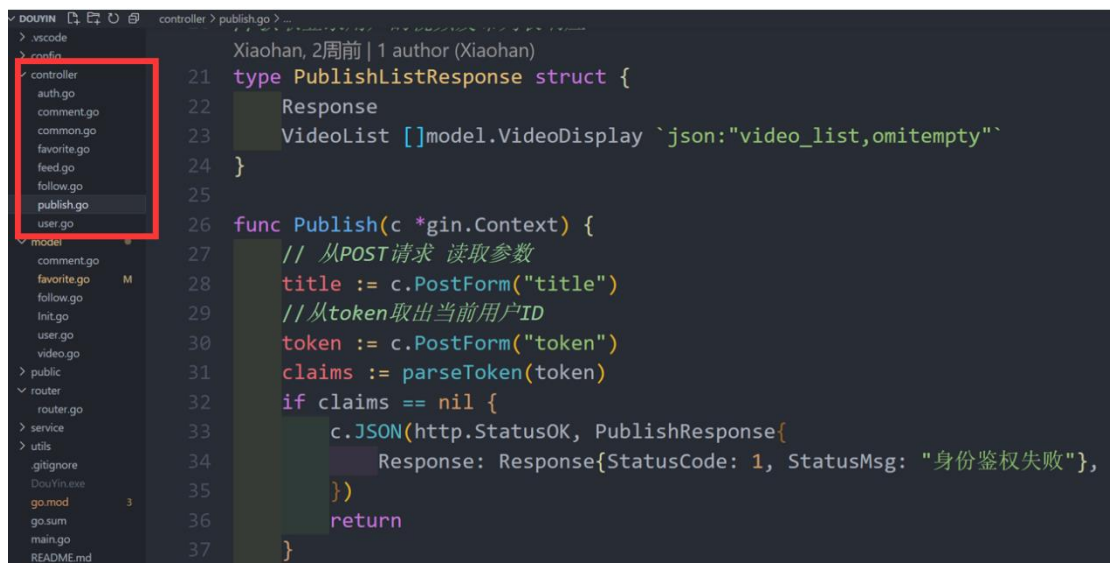
POST 请求，然后交给后续 controller 层的各个处理函数进行处理请求，最后返回对应的结果或者错误信息。

```
func InitRouter(r *gin.Engine) {      Xiaohan, 2周前 • baseAPI基本调通0526 ...
    // public directory is used to serve static resources
    r.Static("/static", "./public")

    apiRouter := r.Group("/douyin")

    // basic apis
    apiRouter.GET("/feed/", controller.Feed)
    apiRouter.GET("/user/", controller.UserInfo)
    apiRouter.POST("/user/register/", controller.Register)
    apiRouter.POST("/user/login/", controller.Login)
    apiRouter.POST("/publish/action/", controller.Publish)
    apiRouter.GET("/publish/list/", controller.PublishList)
```

3. Controller 层，是项目中专门处理请求，这一层包含各种处理函数，主要作用是解析请求中的参数信息，调用 service 层进行进一步处理得到对应的结果，并构造 HTTP 响应。



The screenshot shows a VS Code editor with a project structure on the left and a Go file on the right. The project structure includes folders like 'controller', 'model', 'public', 'router', 'service', and 'utils'. The 'controller' folder is expanded, showing files like 'auth.go', 'comment.go', 'common.go', 'favorite.go', 'feed.go', 'follow.go', 'publish.go', and 'user.go'. The 'publish.go' file is selected. The code in the editor shows the definition of the 'PublishListResponse' struct and the 'Publish' function. The 'Publish' function is a POST handler that takes a 'gin.Context' and returns a 'PublishResponse'.

```
21 type PublishListResponse struct {
22     Response
23     VideoList []model.VideoDisplay `json:"video_list,omitempty"`
24 }
25
26 func Publish(c *gin.Context) {
27     // 从POST请求 读取参数
28     title := c.PostForm("title")
29     // 从token取出当前用户ID
30     token := c.PostForm("token")
31     claims := parseToken(token)
32     if claims == nil {
33         c.JSON(http.StatusOK, PublishResponse{
34             Response: Response{StatusCode: 1, StatusMsg: "身份鉴权失败"},
35         })
36     }
37     return
```

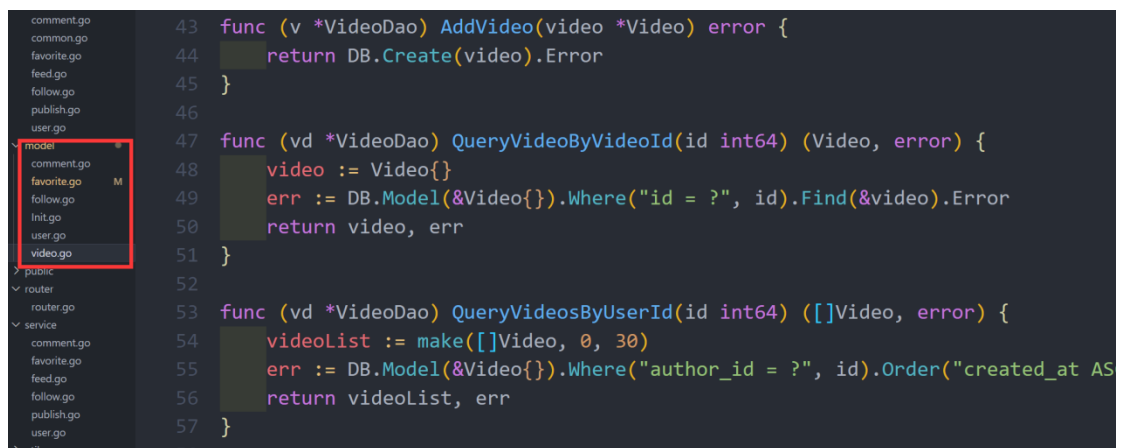
4. service 层，用于进一步处理请求，在这一层需要向上一层屏蔽处理的细节，返回需要的数据信息；此外处理过程中会涉及到多个数据表的读取和修改，因此需要统筹调用 model 层的函数，取用数据库中的重要信息。进行复杂处理，包括异常处理、数据整合之后将数据

打包返回。



```
17 func (p *PublishService) PublishList() ([]model.VideoDisplay, error) {
18     videolist, err := model.NewVideoDao().QueryVideosByUserId(p.Id)
19     if err != nil {
20         return nil, err
21     }
22     // 获得作者信息
23     var userInfo *model.UserInfo
24     videoDisplayList := make([]model.VideoDisplay, 0, 30)
25     userInfo, err = model.NewUserDao().QueryUserById(p.Id)
26     if err != nil {
27         return nil, err
28     }
29 }
```

5. model 层，对数据库进行操作，进行数据库的增删改查，向 service 层返回从数据库中取回的信息。



```
43 func (v *VideoDao) AddVideo(video *Video) error {
44     return DB.Create(video).Error
45 }
46
47 func (vd *VideoDao) QueryVideoByVideoId(id int64) (Video, error) {
48     video := Video{}
49     err := DB.Model(&Video{}).Where("id = ?", id).Find(&video).Error
50     return video, err
51 }
52
53 func (vd *VideoDao) QueryVideosByUserId(id int64) ([]Video, error) {
54     videoList := make([]Video, 0, 30)
55     err := DB.Model(&Video{}).Where("author_id = ?", id).Order("created_at AS
56     return videoList, err
57 }
```

6. public 文件，这里进一步存放 cover 和 video 文件夹，分别存放生成的视频封面和用户上传的视频原文件。在项目中设置 public 为静态资源目录，因此当客户端使用 url 定位到对应的文件信息时，便可在客户端观看封面和视频数据信息。

7. utils 文件，当中存放生成封面的工具函数，在处理上传视频请求过程中调用，传入对应的视频路径、封面存放路径和选取的帧数，即可得到对应的封面图片。

5.2 项目运行

演示视频链接 [展示视频.mp4 - 飞书文档 \(feishu.cn\)](#)

这一部分将会展示项目运行的截图，并作简单的描述讲解，更详细的操作将会在项目录屏当中进行展示。

1. 下面展示的是进行登录和注册的操作，当注册之后便会自动登录并展示最新的视频信息；同样当用户成功登录后也会展示最近的视频。



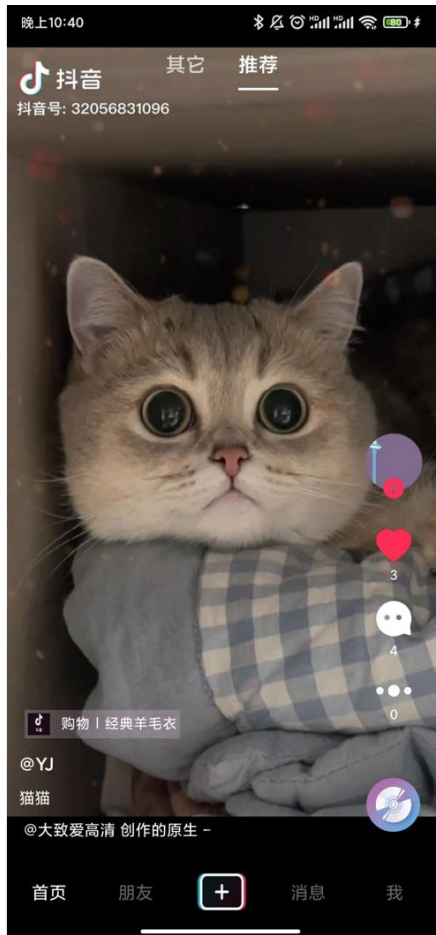
2. 下面展示了登录用户信息页面，可以看到当前用户发布的作品列表和点赞列表，列表中信息包含各个视频的封面和对应点赞数量情况。



3. 下面展示用户信息页的关注列表和粉丝列表，在对应用户后可以选择点击是否关注或者取消关注。



4. 下面展示视频页面，登录用户可以对视频进行点赞和评论，在评论界面可以看到当前视频的所有评论。



六 项目总结

6.1 总结

本次抖音项目实践，成功实现了简易版抖音，在测试过程中能够通过抖声 APP 在手机上进行使用，使用体验与实际使用抖音 APP 感受类似。能够让使用者方便快捷的观看最新的视频，进行注册和登录，并且在登录状态下对用户进行关注，对视频进行点赞和评论，以及上传视频。

在项目实践过程中，小组成员熟悉使用 git 进行合作开发，将 git 知识运用到了实战，并且促进小组合作交流。

6.2 展望

在实际的项目当中，为了能够快速响应用户的请求，并且扩大请求并发量，应该增加 Redis 的使用。

在用户请求后端时，会从数据库中取出相关信息，但这个过程会比较慢，如果将访问的数据放入缓存，这样在下一次访问这些数据时就可以直接从缓存中获取。此外直接操作缓存能够承受的请求是远远大于访问数据库，通过将数据库部分数据转移到缓存，能够实现高并发功能。

在本次项目中，在评论功能中增加了 Redis，但在其他点赞等功能还没使用上 Redis，因此在之后的学习中希望能掌握 Redis 的相关知识和使用，为以后的项目打好坚实基础。