**巅峰的20岁-dousheng-项目介绍文档**

**1 项目提交**

**1.1 项目仓库**

项目以gitee为代码托管平台，连接Goland协作开发，开发完成后共享到github平台。

github仓库如下：

**[该类型的内容暂不支持下载]**

gitee仓库如下：

**[该类型的内容暂不支持下载]**

**1.2 演示视频**

|  |
| --- |
| 演示操作步骤：   1. 基础接口   以不存在用户登录（登录失败）-->注册用户（注册成功）-->刷新视频（刷到其他用户已发布视频）-->发布视频（发布成功）-->刷新视频（刷到刚刚发布的视频）   1. 扩展接口I   点赞视频（成功点赞）-->观看点赞列表（已有刚刚点赞视频）-->取消点赞视频（取消成功）-->观看点赞列表（已无视频）-->查看评论列表（无评论）-->评论视频（评论成功）-->删除评论（已无评论）   1. 扩展接口II   关注作者（关注成功）-->查看关注列表（已有刚刚关注的作者）-->查看关注者粉丝列表（存在登录账户）-->取消关注（取消成功）-->查看关注列表（已无关注）-->查看关注者粉丝列表（已无粉丝） |

* **录屏中，输入密码期间手机会黑屏保护，属于正常现象！！**

**[peak20.mp4]**

**视频1 演示视频**

视频备用链接：http://rd5met9ed.hn-bkt.clouddn.com/video/6014959579190923264.mp4

**1.3 整体评价**

|  |
| --- |
| 本项目已完成基础接口、扩展接口I、扩展接口||所有功能项，代码结构清晰，符合编码规范，并采用Redis与MySQL混合存储、数据库表去冗余设计等方式提高性能，同时关注、喜爱等相关操作均验证用户token，考虑安全问题，具体如下表所示： |

**表 1 项目完成情况表**

|  |  |  |
| --- | --- | --- |
| **评价项** | **完成情况** |  |
| **功能实现** | 基础接口 | * 未登录用户也可以刷抖音 * 按投稿时间倒序推出（PublishTime int64 `json:"publish\_time"`） * 登录用户可以投稿，查看自己的基本信息和投稿列表 |
| 扩展接口I | * 登录用户可以点赞、评论，在个人主页能够查看点赞视频列表。 |
| 扩展接口III | * 登录用户可以关注用户，查看本人的关注数和粉丝数、关注列表和粉丝列表 |
| **代码质量** | * **代码层次清晰**分为 * common（通用的返回消息和错误号构造） * config（Redis与MySQL配置） * database（数据库池化） * controller（解析请求并转发给service） * model（Video、VideoInfo、User、UserInfo等数据结构） * router（路由转发给controller） * service（服务处理函数） * logger（日志） * **函数编写规范** * 函数命名可以直接明了获得函数功能，如“GetCommentByVideoID”未根据video\_id获取comment * 高复用代码独立成函数，如“GetUserInfoListByIDs”根据id获取用户信息 * 单一函数长度不超过200行，函数功能明确，如关注功能中，“获取两人是否关注”、“获取关注数”、“获取粉丝数”均采用不同函数实现，没有杂糅在同一函数中 |  |
| **服务性能** | * **Redis中Zset优化关注与喜爱数据存储** * 使用Redis中有序且不重复的高性能结构Zset存储用户关注以及用户喜爱视频数据，避免大量数据造成负担，如假设有一千个用户，平均每人有一万个粉丝关注，用mysql就是千万级别的记录数 * **MySQL数据库表去冗余设计** * 评论表中不存储全部用户信息，仅存储用户id * 用户表中不存储用户的关注、被关注、关注数、用户名信息，仅存储ID与密码 * **以上信息通过在拉取时，通过Redis配合，动态的查询返回，能够在提高存储效率的同时，有效避免存储与拉取时关注关系不一致的问题** * **对象存储** * 使用七牛云对象存储Kodo，高可靠，灵活应对大流量的业务场景 * **docker部署** * 使用docker快速部署Redis、MySQL服务，整合相关依赖 * **视频封面获取** * 定位视频第一秒作为视频封面 |  |
| **安全可靠** | * **权限验证** * 相关函数执行时均会验证token * 关注、喜爱信息获取时会再次根据token匹配用户信息，避免用户信息不一致 * **Uber-go zap实现日志记录** * 使用Uber-go zap实现日志记录，处理历史数据、诊断问题的追踪 * 同时提供了结构化日志记录和printf风格的日志记录 |  |

**1.4 任务分工**

|  |
| --- |
| 本项目团队共5人：王洪涛、文峥、黄春林、王先华、蒋刚，团队分工如下表所示： |

**表 2 团队分工情况表**

|  |  |  |  |
| --- | --- | --- | --- |
| **任务类别** | **成员名称** | **负责接口名称** |  |
| **编码** | @王洪涛 | * 基础接口 * ~~视频流接口-/douyin/feed~~ * ~~用户注册-/douyin/user/register/~~ * ~~用户登录-/douyin/user/login/~~ * 扩展接口II * ~~关注操作-/douyin/relation/action/~~ * ~~关注列表-/douyin/relation/follow/list/~~ * ~~粉丝列表-/douyin/relation/follower/list/~~ |  |
| @文峥 | * 扩展接口I * ~~赞操作-/douyin/favorite/action/~~ * ~~点赞列表-/douyin/favorite/list/~~ * ~~评论操作-/douyin/favorite/list/~~ * ~~评论列表- /douyin/favorite/list/~~ |  |
| @黄春林 | * 基础接口 * ~~视频流接口-/douyin/feed~~ * ~~投稿接口-/douyin/publish/action/~~ * ~~发布列表-/douyin/publish/list/~~ |  |
| **测试** | @王先华 | * 扩展接口I * ~~赞操作-/douyin/favorite/action/~~ * ~~点赞列表-/douyin/favorite/list/~~ * ~~评论操作-/douyin/favorite/list/~~ * ~~评论列表- /douyin/favorite/list/~~ | * 基础接口 * ~~视频流接口-/douyin/feed~~ * ~~用户注册-/douyin/user/register/~~ * ~~用户登录-/douyin/user/login/~~ * ~~投稿接口-/douyin/publish/action/~~ * ~~发布列表-/douyin/publish/list/~~ |
| @蒋刚 | * 扩展接口II * ~~关注操作-/douyin/relation/action/~~ * ~~关注列表-/douyin/relation/follow/list/~~ * ~~粉丝列表-/douyin/relation/follower/list/~~ |

**2 项目优势**

**2.1 Redis与MySQL结合存储**

使用Redis中有序且不重复的高性能结构Zset存储用户关注以及用户喜爱视频数据，避免大量数据造成负担

|  |
| --- |
| 如假设有一千个用户，平均每人有一万个粉丝关注，用mysql就是千万级别的记录数，而使用Zset将避免此种负担 |

**2.2 数据库表去冗余设计**

* 评论表中不存储全部用户信息，仅存储用户id

|  |
| --- |
| Go type Comment struct {  VideoID int64 `json:"video\_id"`  Content string `json:"content"` // 评论内容  CreateDate string `json:"create\_date"` // 评论发布日期，格式 mm-dd  CommentID int64 `gorm:"primaryKey autoIncrement" json:"comment\_id"` // 评论id  UserID int64 `json:"user\_id"` // 评论用户信息 }  type CommentInfo struct {  VideoID int64 `json:"video\_id"`  Content string `json:"content"` // 评论内容  CreateDate string `json:"create\_date"` // 评论发布日期，格式 mm-dd  CommentID int64 `json:"comment\_id"` // 评论id  User UserInfo `json:"user"` // 评论用户信息 } |

* 用户表中不存储用户的关注、被关注、关注数、用户名信息，仅存储ID与密码

|  |
| --- |
| Go type User struct {  ID int64 `gorm:"primaryKey autoIncrement" json:"user\_id"`  Username string `gorm:"uniqueIndex type:varchar(32)" json:"username"` // required，最长32个字符  Password string `gorm:"type:varchar(32)" json:"password"` // required，最长32个字符 }  type UserInfo struct {  Id int64 `json:"id,omitempty"`  Name string `json:"name,omitempty"`  FollowCount int64 `json:"follow\_count,omitempty"`  FollowerCount int64 `json:"follower\_count,omitempty"`  IsFollow bool `json:"is\_follow,omitempty"` } |

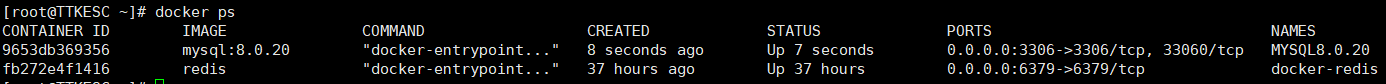
* **以上信息通过在拉取时，通过Redis配合，动态的查询返回，能够在提高存储效率的同时，有效避免存储与拉取时关注关系不一致的问题**

**2.3 对象存储**

使用七牛云对象存储Kodo，高可靠，灵活应对大流量的业务场景

**2.4 docker部署**

使用docker快速部署Redis、MySQL服务，整合相关依赖



**2.5 视频封面截取**

定位视频第一秒作为视频封面

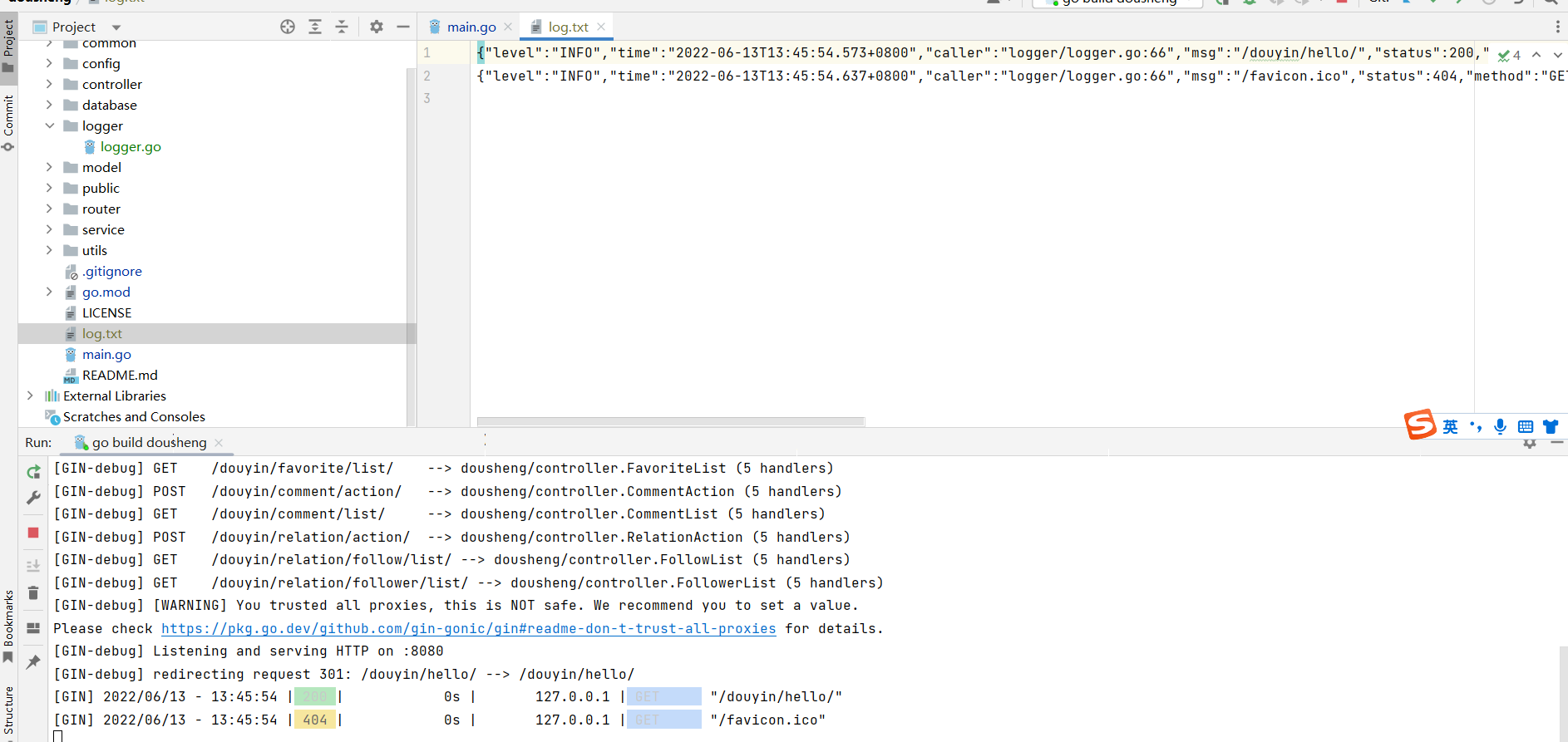
|  |
| --- |
| Go //视频封面start coverName := filename + "." + "jpg" //封面的文件名 coverFolderName := "cover" //七牛云中存放图片的目录名。用于与文件名拼接，组成文件路径 coverKey := coverFolderName + "/" + coverName //封面的访问路径，我们通过此路径在七牛云空间中定位封面 saveJpgEntry := base64.StdEncoding.EncodeToString([]byte(bucket + ":" + coverKey)) putPolicy.PersistentOps = "vframe/jpg/offset/1/w/534/h/949|saveas/" + saveJpgEntry //取视频第1秒的截图 |

**2.6 权限验证**

相关函数执行时均会验证token；关注、喜爱信息获取时会再次根据token匹配用户信息，避免用户信息不一致

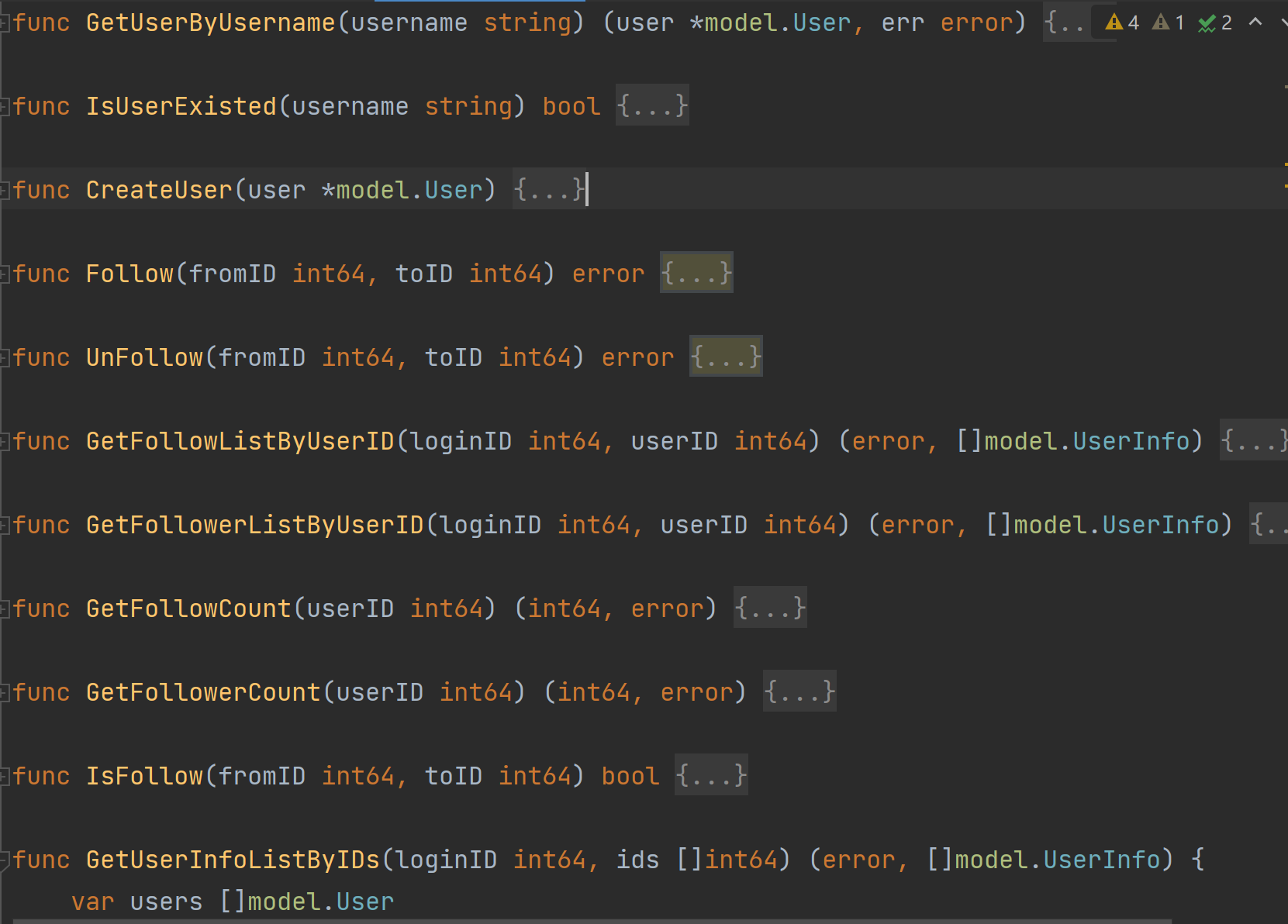
**2.7 Uber-go zap日志记录**

使用Uber-go zap实现日志记录，处理历史数据、诊断问题的追踪；同时提供了结构化日志记录和printf风格的日志记录



**2.8 编码清晰可扩展**

* **代码层次清晰**分为
* common（通用的返回消息和错误号构造）
* config（Redis与MySQL配置）
* database（数据库池化）
* controller（解析请求并转发给service）
* model（Video、VideoInfo、User、UserInfo等数据结构）
* router（路由转发给controller）
* service（服务处理函数）
* utils（response封装）
* **函数编写规范**
* 函数命名可以直接明了获得函数功能，如“GetCommentByVideoID”未根据video\_id获取comment
* 高复用代码独立成函数，如“GetUserInfoListByIDs”根据id获取用户信息
* 单一函数长度不超过200行，函数功能明确，如关注功能中，“获取两人是否关注”、“获取关注数”、“获取粉丝数”均采用不同函数实现，没有杂糅在同一函数中



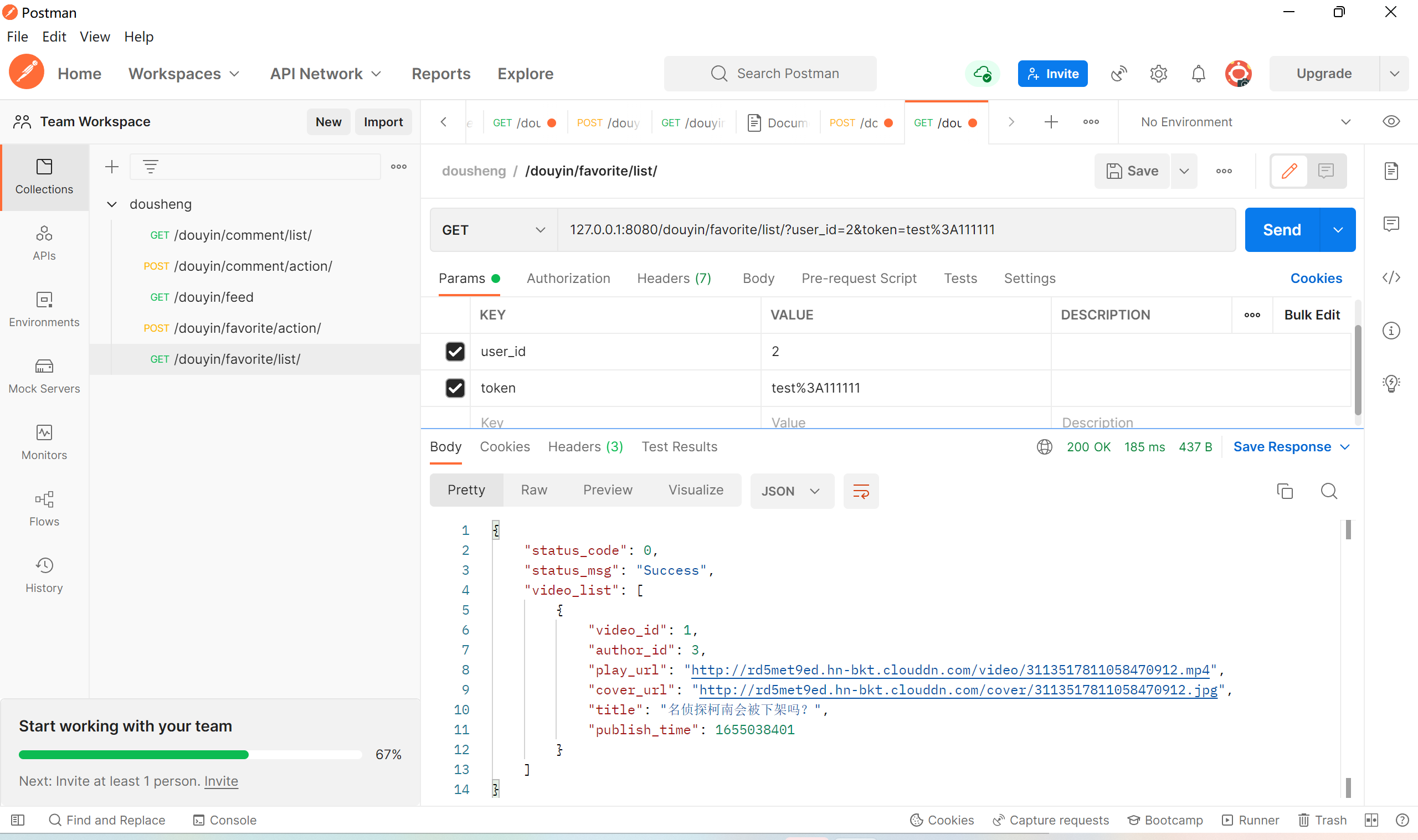
**3 项目结构**

**3.1 技术框架**

开发：gin+gorm+七牛云Kodo+Uber-go zap

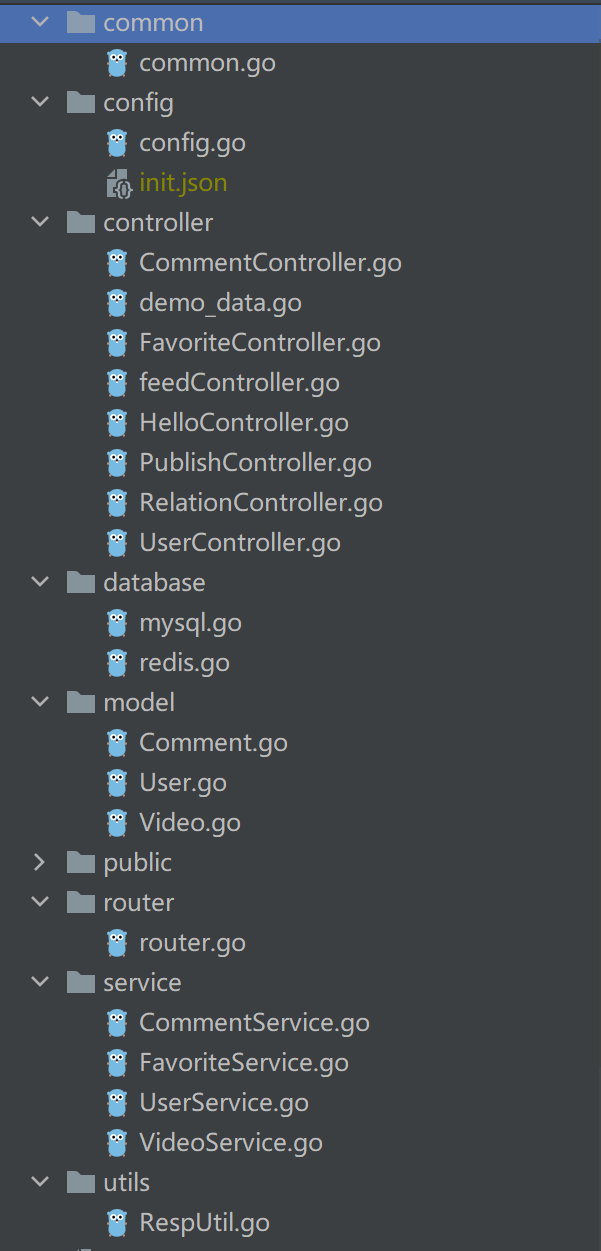
数据库：Redis+MySQL

测试：postman



**3.2 整体结构**

|  |
| --- |
| 代码目录结构为：   * common（通用的返回消息和错误号构造） * common.go * config（Redis与MySQL配置） * config.go（结构信息） * init.json（配置文件） * controller（解析请求并转发给service） * CommentController.go（评论、获取评论列表） * FavoriteController.go（点赞、查看点赞视频列表） * feedController.go（拉取视频） * PublishController.go（发布视频） * RelationController.go（关注、粉丝数、查看相关列表） * UserController.go（登录与注册） * HelloController.go（登录或注册成功） * database（数据库池化） * mysql.go * redis.go * model（Video、VideoInfo、User、UserInfo等数据结构） * Comment.go（存储在MySQL中的评论信息以及Response返回的评论信息） * User.go（存储在MySQL中的用户信息以及Response返回的用户信息） * Video.go（存储在MySQL中的视频信息） * router（路由转发给controller） * router.go * service（服务处理函数） * CommentService.go * FavoriteService.go * UserService.go * VideoService.go * logger（日志） * logger.go |



**4 部分调试问题总结**

* **GO服务运行apk文件**

电脑连接校园网，电脑开移动热点，手机连接电脑热点，手动配置这个WiFi代理，配置成电脑ip+端口。手机登录注册电脑可收到消息。

*我们通常使用localhost或者127.0.0.1来访问本机的Web服务，但是如果我们在Android模拟器中也采用同样的地址来访问，Android模拟器将无法正常访问到我们的服务，这是为什么呢？我们可以这样来理解：Android的底层是Linux kernel，包括Android本身就是一个操作系统，因此，这时我们在模拟器的浏览器中输入的localhost或127.0.0.1所代表的是Android模拟器（Android*[*虚拟机*](https://so.csdn.net/so/search?q=%E8%99%9A%E6%8B%9F%E6%9C%BA&spm=1001.2101.3001.7020)*），而不是你的电脑，明白了吗？这就是为什么你在模拟器中使用localhost时会报“Web page not available”的原因。*

*那到底要如何才能访问到本地电脑上的Web应用呢？在Android中，默认将我们本地电脑的地址映射为10.0.2.2，因此，只需要将原先的localhost或者127.0.0.1换成10.0.2.2，就可以在模拟器上访问本地计算机上的Web资源了。*

* **ShouldBind**

能够基于请求的不同，自动提取JSON、form表单和QueryString类型的数据，并把值绑定到指定的结构体对象

* **电脑adb连接木木模拟器**

环境遍历配置C:\Program Files (x86)\MuMu\emulator\nemu\vmonitor\bin

adb connect 127.0.0.1:7555

adb devices

木木模拟器连接服务器地址10.0.2.2:8080

* **模拟器有网络，可以访问七牛云网址看视频。APP里看不到视频**

数据库里的存url时没加http://

* **AutoMigrate**

可用于建表。增加struct字段时类似mongo自动加列。很灵活很方便。

|  |
| --- |
| Go db.AutoMigrate(&Product{}) |

* **允许gin传递0值的办法为**

|  |
| --- |
| Go type GetInferenceTaskRule struct {  Page int `form:"page" binding:"required"`  Size int `form:"size" binding:"required"`  IsOnMyself \*int `form:"is\_on\_myself" binding:"required"` } |

*想要验证通过也很简单，我们只要把类型定义为指针类型即可接受零值。接收到的这个\*int怎么用呢？*

*pageonmyself := \*form.IsOnMyself*

* **可以先用postman测试接口，查看服务端报错，如果postman没有问题，但实现不能达到预期，可能是参数设置的问题**