

一、项目介绍 [陈宇航，李浩腾]

背景

在现代，生活节奏不断加快，人们为了生活中的种种事情而疲于奔命，有些人辛苦了一整天，第二天回想起来这一天的时候，却什么都记不起来。因此，我们的项目“随心记”就由此产生，它帮助人们在有空时，用文字和图片将自己生活中的碎片整理起来，就像是小时候那本珍爱的日记本。用户可以为自己的日记设置密码、提醒以及是否公开，当想回忆日记内容时候可以按日期、关键字进行搜索。同时，那些公开的日记形成了一个日记圈，用户可以在里面找到和自己兴趣相投的用户。

项目需求分析

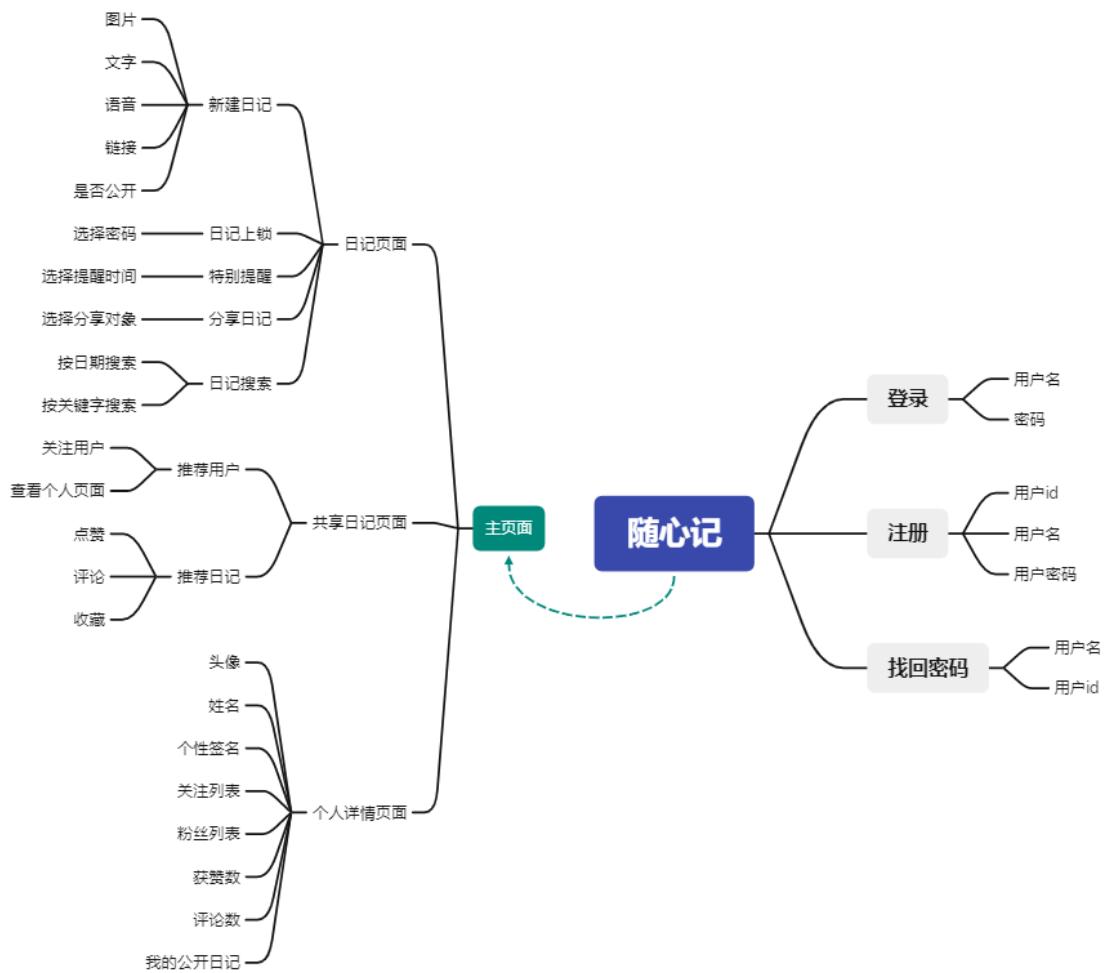
1.项目定位：

随心记项目主要定位于那些有写日记习惯的人群，为他们提供了一个简洁高效的软件去完成自己记日记的需求，同时可以对日记进行上锁、提醒、搜索等操作，如果愿意将自己的生活碎片分享给大众，可以将自己的日记设为公开，分享到热门日记圈子里，来满足部分对社交有需求的用户群体。

2.项目需求汇总：

序号	使用场景	需求提炼	对应功能	需求类型
1	有记手账、写日记习惯的人群，但不能将记事本随身携带	需要一款能用来记事的APP	日记	基本
2	希望将自己的生活分享给别人，找到兴趣相投的朋友	可以将生活点滴分享给他人	日记圈子	基本
3	希望打造自我空间，增加隐私	日记私密上传	密码锁	基本
4	时常会忘记生活中的琐事，可以提醒用户	在设定的时间提醒用户	定时提醒	基本
5	喜欢定期整理自己过往生活碎片	搜索一段时间内自己写的日记	按时间搜索	基本
6	希望首页推荐看上去舒适直观	卡片式布局	排版优化	期望

3.项目功能结构：



4. 功能需求清单：

需求分类		需求说明	优先级
登录界面	已登录状态	用户可以进行APP内所有操作	高
	未登录状态	用户仅可以浏览页面	高
	帐号注册	用户填写ID、用户名、密码进行注册	高
	帐号登录	用户输入用户名和密码进行登录	高
	忘记密码	系统根据ID和用户名确定用户身份	高
日记主页面	新建日记	用户输入日记标题和日记内容	高
	查看日记详情	系统显示详细日记标题和日记内容	高
	编辑日记	用户修改日记标题和日记内容	高
	删除日记	用户可以选择日记进行删除	高
	上锁日记	用户选择日记，设置密码进行上锁	高
	解锁日记	用户填写密码对日记解锁	高
	按日期搜索日记	用户选择日期搜索， 系统显示对应时间段的日记	中
	按关键字搜索日记	用户输入关键字搜索， 系统显示含有关键字的日记	中
	按时提醒	用户设置提醒时间， 系统在设定的时间提醒用户	中
日记圈页面	点赞日记	用户对日记进行点赞	中
	评论日记	用户对日记进行评论	中
	查看热门日记详情	用户查看热门的详情	中
	查看用户详情	用户点击查看他人详情	中
	关注用户	用户点击关注他人	中
个人页面	创建个人详情资料	用户首次登录APP，输入姓名、 签名、头像、性别创建资料	高
	编辑个人详情资料	用户修改个人详情资料	高
	查看关注列表	用户点击查看关注的人	低
	查看粉丝列表	用户点击查看自己的粉丝	低
	查看点赞列表	用户点击查看自己点赞过的日记	低
	查看评论列表	用户点击查看自己评论过的日记	低

计划和分工

1. 开发计划

整体开发计划遵循课程的流程，先对项目功能需求进行分析，整理得到功能思维导图，之后根据功能确定页面的设计，使用墨刀UI进行原型设计，再之后是进行接口设计，来确保前后端开发的一致性，后端部分写好测试，以备每个接口出错时及时找到问题所在。

2. 项目分工

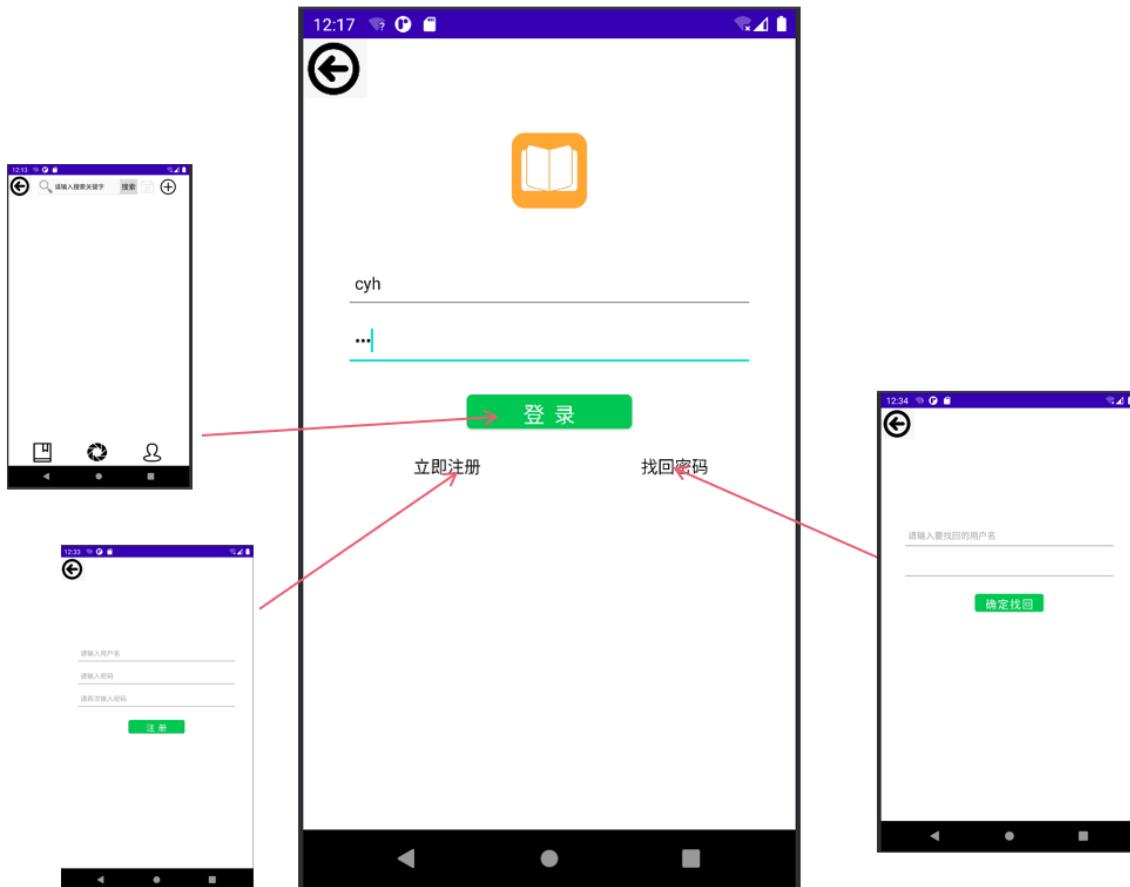
陈宇航：前端用户登录模块、日记模块、个人详情模块的功能实现和原型设计，后端全部内容，数据表设计，API设计，前、后端测试，前后端联调

李浩腾：前端日记圈子模块、个人详情模块的功能实现和原型设计，前端测试，前后端联调

二、界面原型设计 [陈宇航, 李浩腾]

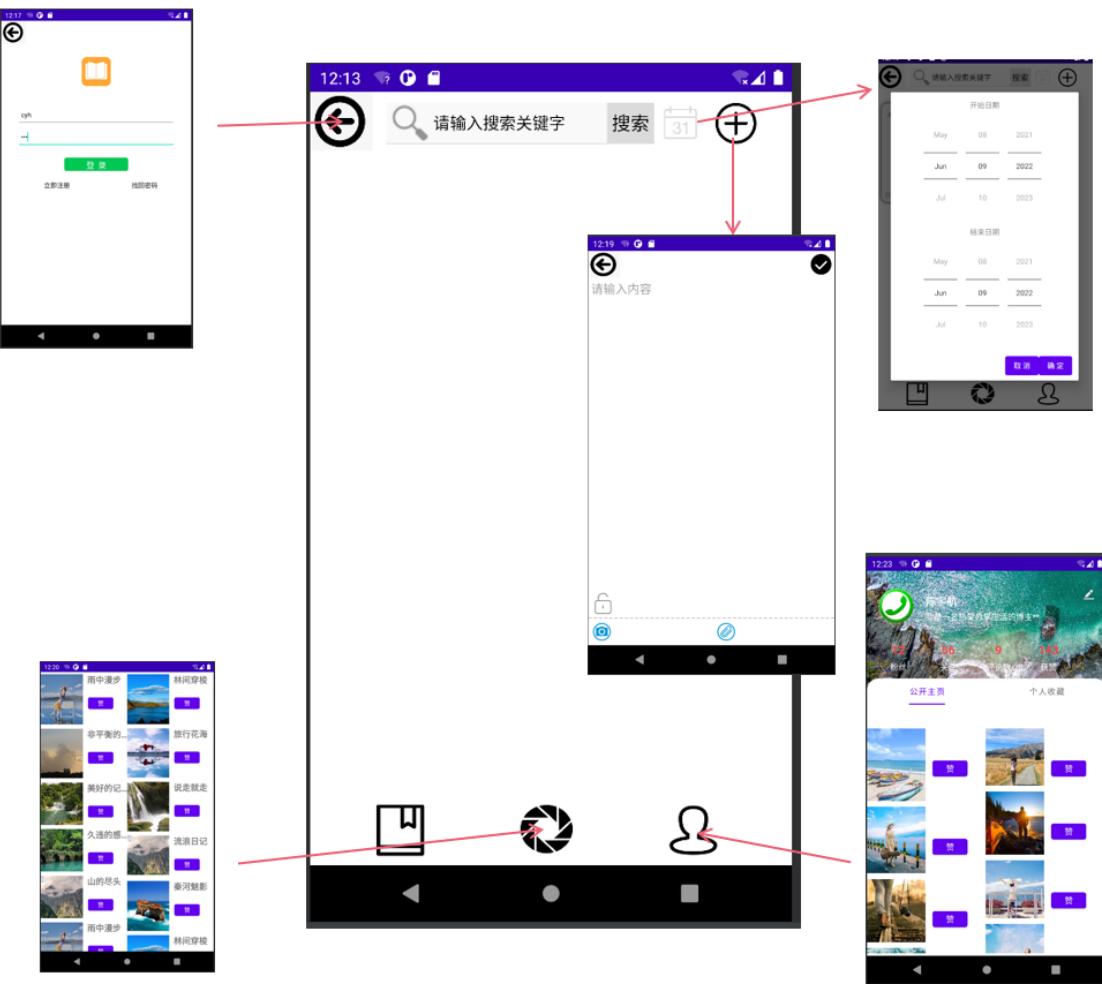
一. 登录界面

该处为登录页面，点击“立即注册”按钮跳转至注册界面，点击“找回密码”跳转至找回密码界面，登录按钮用于在登录成功后，跳转到系统主界面。

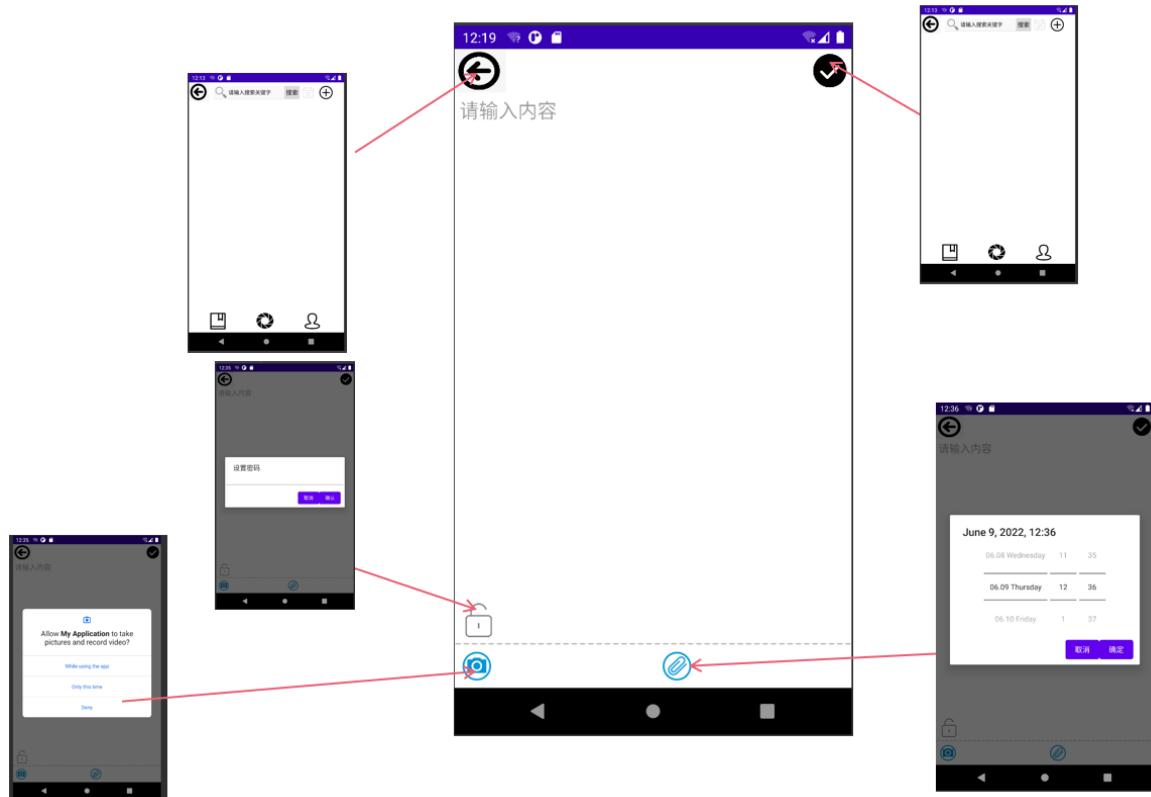


二. 主页面

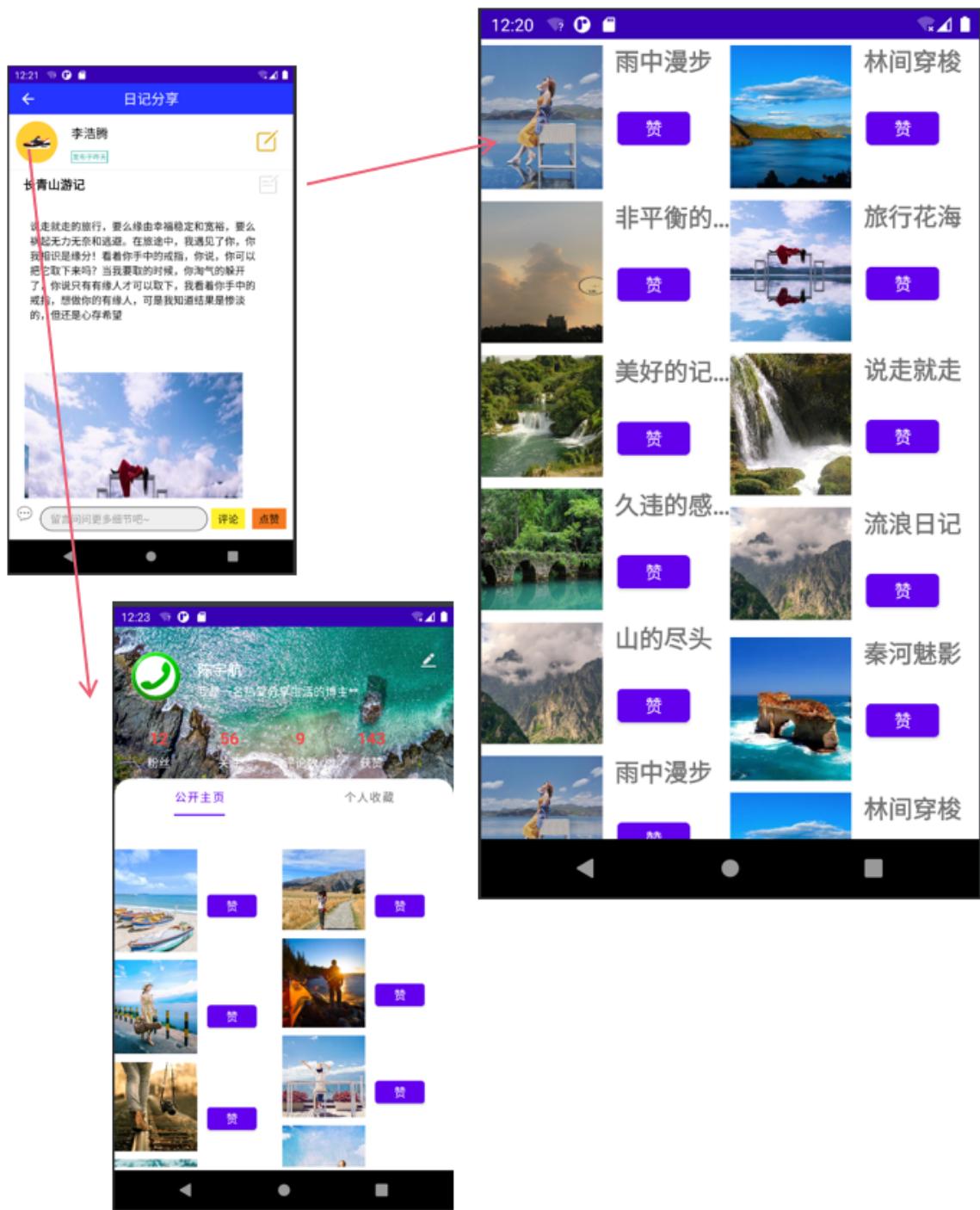
主页面下方由一组导航栏构成，开始默认在日记主界面，中间的可跳转至热门日记圈子界面，最右边的是跳转到个人详情页面。上方的控件用于功能的实现，搜索框的EditText，可以在里面输入文字内容，按下搜索后对日记内容进行筛选，日记框是用来筛选日期，加号箭头跳转到新增日记页面。



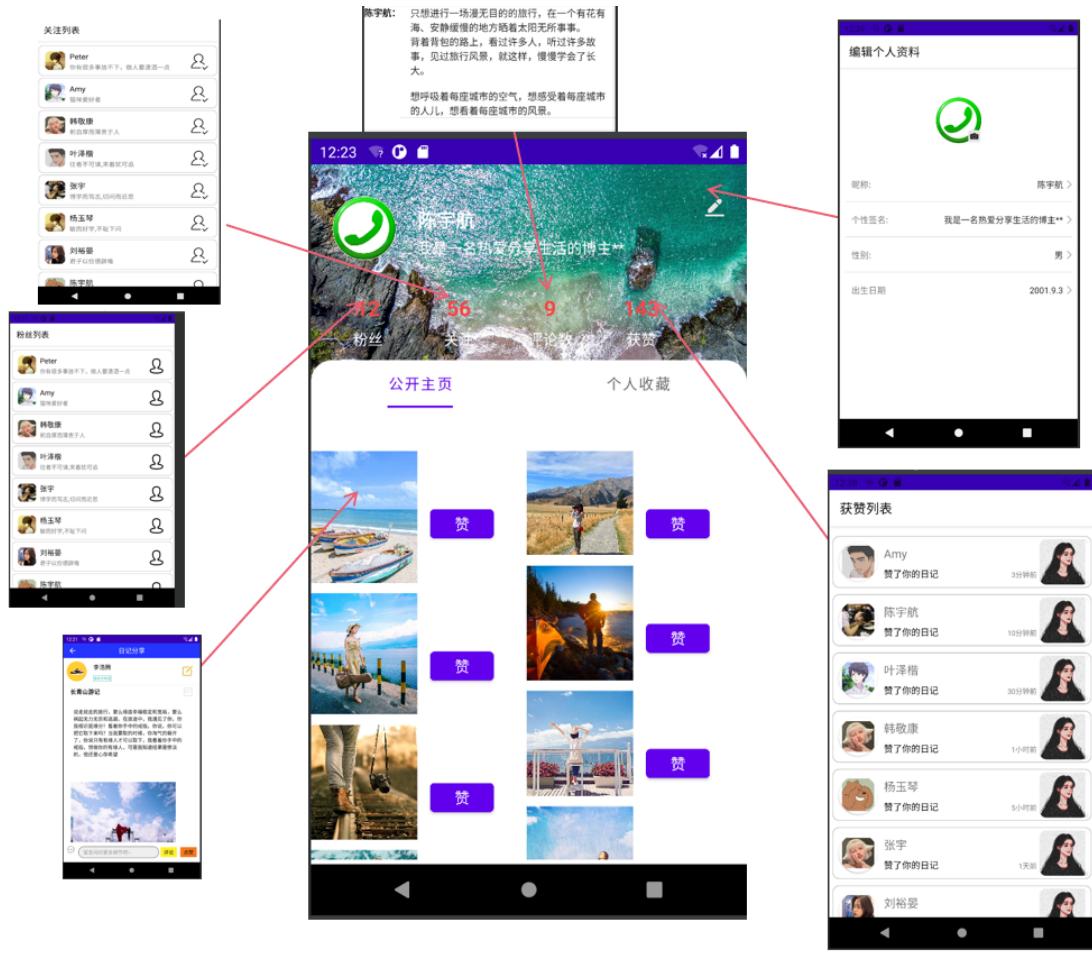
三.新增日记页面



四.日记圈子页面



五.个人详情页面



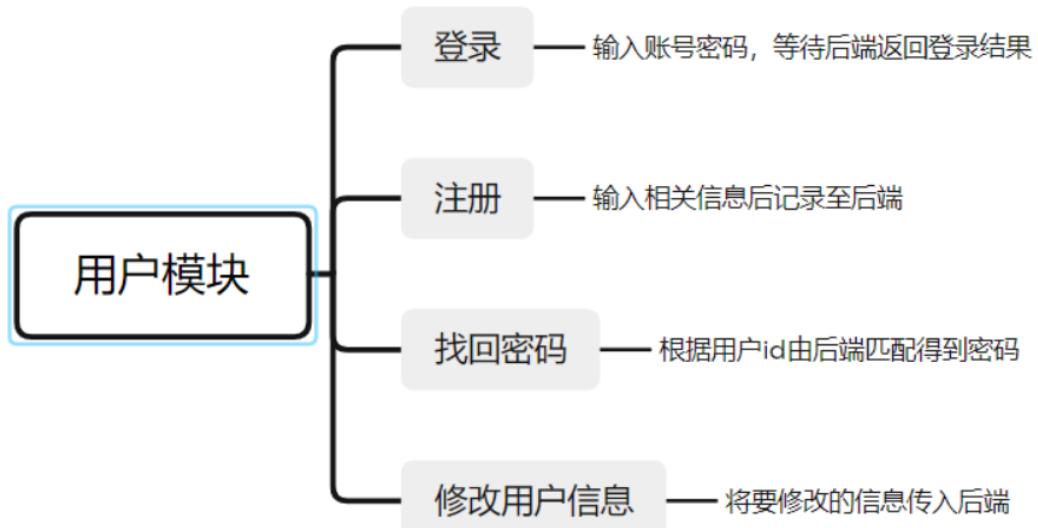
三、系统架构设计 [陈宇航、李浩腾]

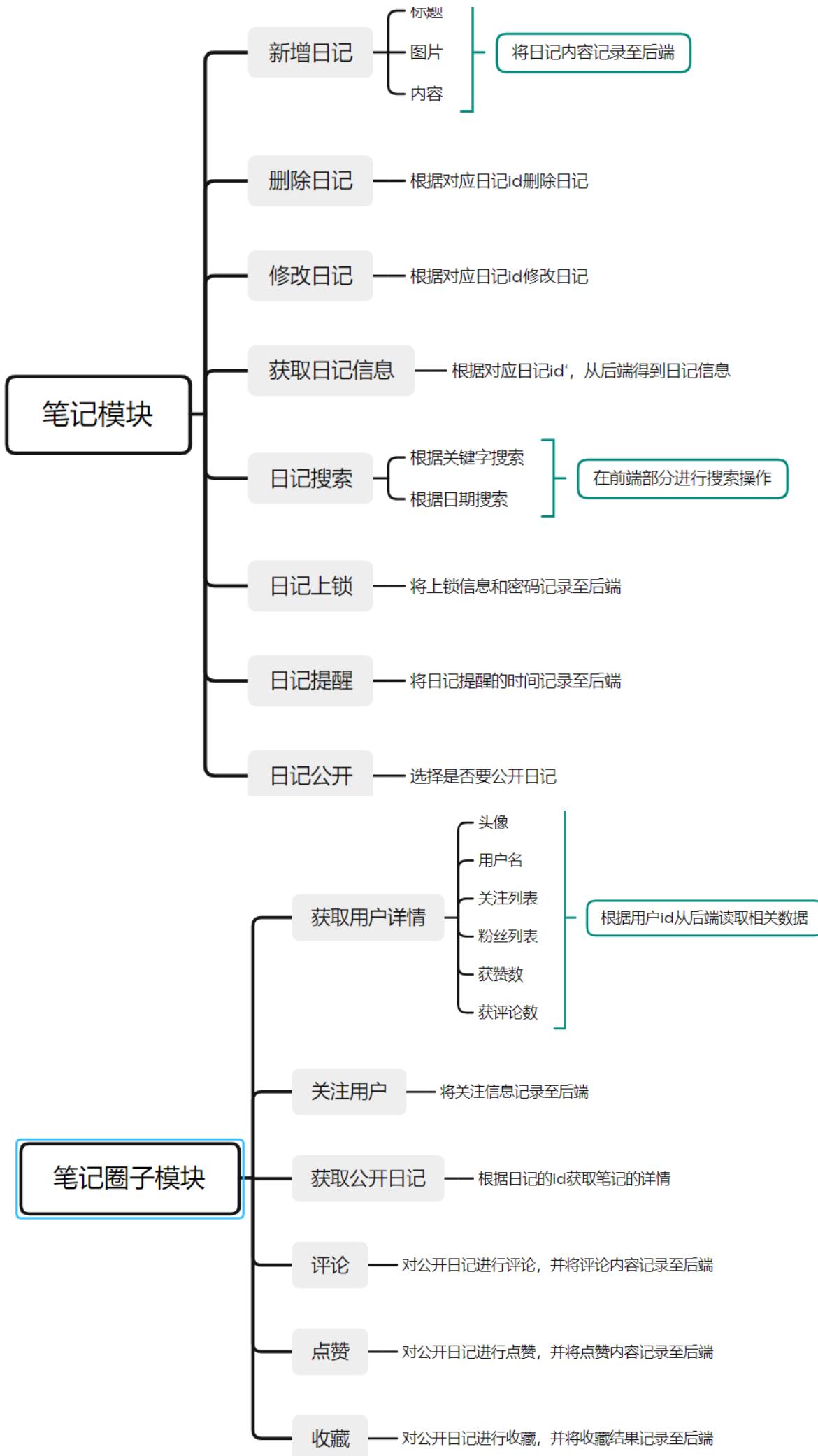
1. 前端架构

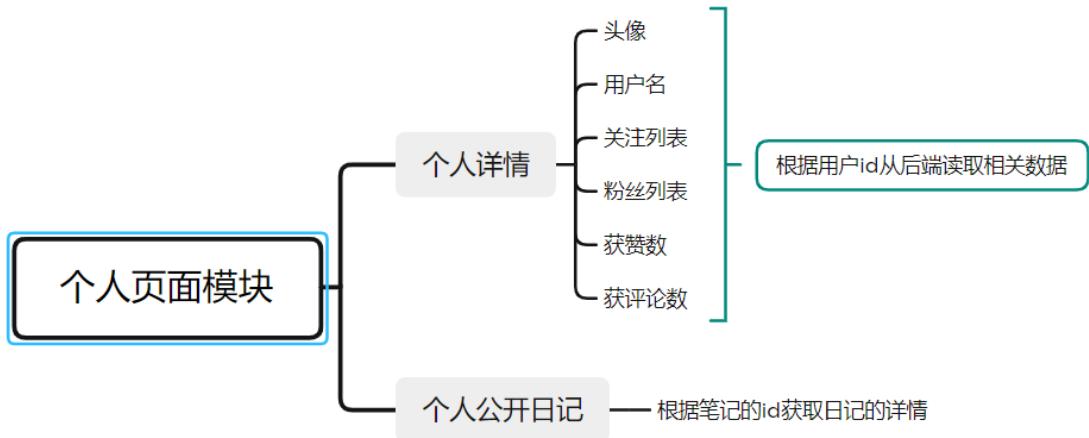
Android构架按照MVC模式开发，分为了Model层、View层和Controller层。Model层是View层和Controller层的中间层，主要负责处理全局数据，保证全局数据的一致；View层是控制应用视图显示的一层，主要作用是显示应用数据；Controller层是MVC架构中最重要的一层，实现了客户端底层的逻辑处理，使得客户端能够平稳运行。

前端可以根据功能分为四个模块，分别是用户管理模块、日记主模块、热门日记圈子模块、个人详情模块

具体的模块内部的功能以及设计可见架构图：



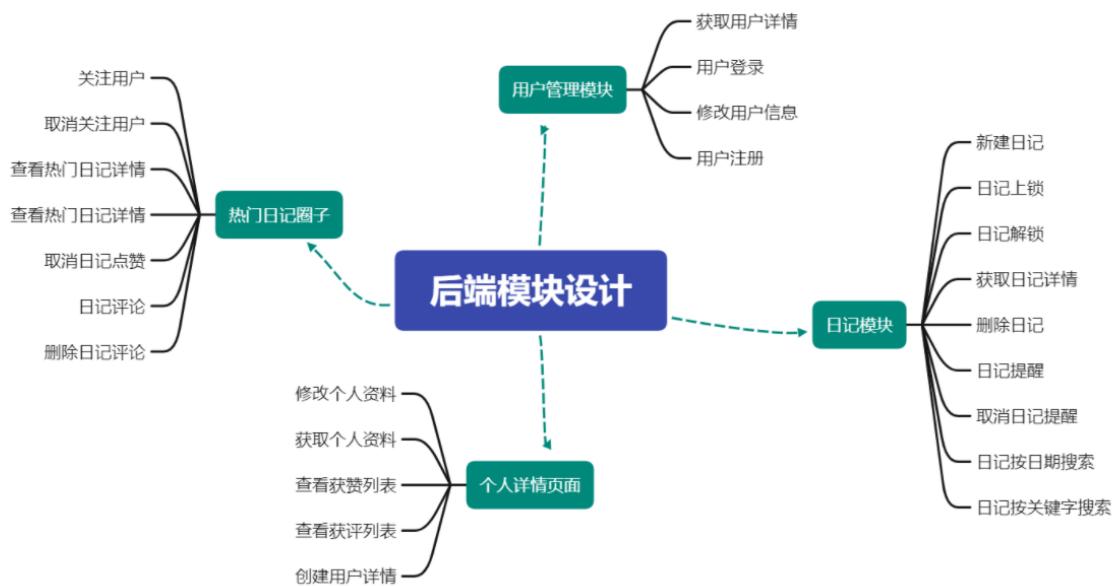




2. 后端架构

后端采用的是django技术，依托MVT框架进行开发

后端主要也是根据整体的功能分为四个模块：用户管理模块、日记模块、热门日记圈子模块、个人详情模块，具体每一个模块的信息和具体设计如下图：

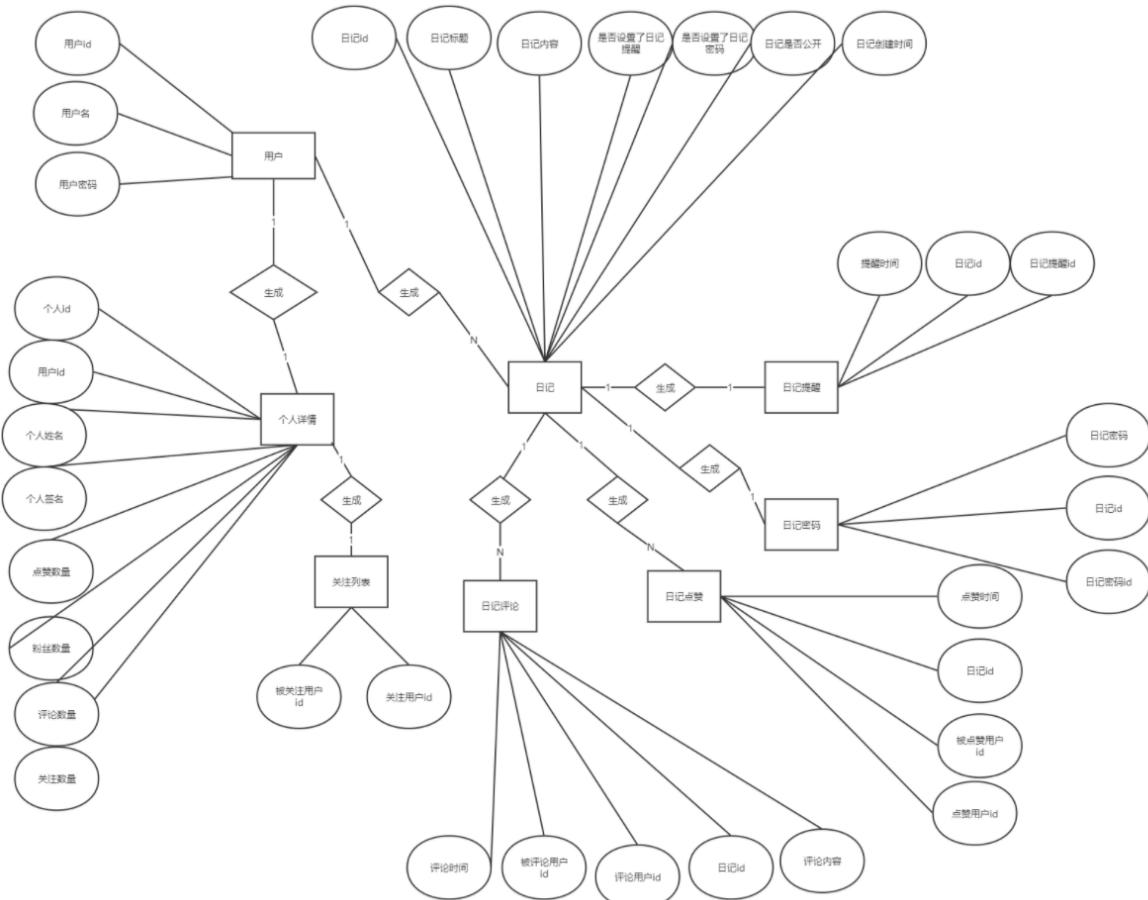


3. API部分

API部分采用APIFox进行设计，每一个接口都和后端对应url一一对应（接口名、参数名），使得开发时可以非常便利的知道接口信息

4. 数据库

数据库部分选择django后端自带的SQLite+ORM，原因是这两个技术都是后端本身自带的，这样在不同组员协作开发时，只需将从github上下载项目即可直接运行，不需要再进行环境配置的繁琐操作



5. 技术方案实现

前端: AndroidStudio

后端: Django

数据库: ORM+SQLLite

原型设计: 墨刀UI

接口设计: APIfox

四、API设计 [陈宇航]

该处我选择直接用APIfox生成接口文档，便于更加直观的说明API的设计理念，使用以及响应数据。

APIfox在线分享链接: <https://www.apifox.cn/apidoc/shared-bd557f21-16d7-49ca-b0ba-9237b20a0374>

1. 用户管理模块

(1) GET 用户登录

GET /UserLogin/{username}/{passwd}

用户登录接口主要通过向后端传用户名username、用户密码passwd两个参数，后端在数据库中进行匹配，若存在则返回“登录成功”信息，若失败则返回“登录失败”的信息

请求参数

名称	位置	类型	必选	说明
username	path	string	是	用户名
passwd	path	string	是	用户密码

返回示例

成功

```
{  
    "ret": 0,  
    "msg": "登录成功"  
}
```

失败

```
{  
    "ret": 1,  
    "msg": "登陆失败"  
}
```

返回结果

状态码	状态码含义	说明	数据模型
200	OK	成功	Inline
201	Created	失败	Inline

返回数据结构

状态码 200

名称	类型	必选	约束	说明
» ret	integer	true	none	0代表成功, 1代表失败
» msg	string	true	none	提示性消息

状态码 201

名称	类型	必选	约束	说明
» ret	integer	true	none	0代表成功, 1代表失败
» msg	string	true	none	提示性消息

(2) PUT 修改用户信息

PUT /UserChange

修改用户信息，主要用于对用户名和密码进行修改，其中以用户id作为判别依据，若不存在对应的用户id，则提示错误信息

Body 请求参数

```
{  
  "userid": "string",  
  "username": "string",  
  "passwd": "string"  
}
```

请求参数

名称	位置	类型	必选	说明
body	body	object	否	none
» userid	body	string	是	用户id
» username	body	string	是	用户名
» passwd	body	string	是	用户密码

返回示例

成功

```
{  
  "ret": 0,  
  "msg": "修改成功"  
}
```

失败

```
{  
  "ret": 1,  
  "msg": "用户不存在，请重新输入"  
}
```

返回结果

状态码	状态码含义	说明	数据模型
200	OK	成功	Inline
201	Created	失败	Inline

返回数据结构

状态码 200

名称	类型	必选	约束	说明
» ret	integer	true	none	0代表成功，1代表失败
» msg	string	true	none	提示性消息

状态码 201

名称	类型	必选	约束	说明
» ret	integer	true	none	0代表成功，1代表失败
» msg	string	true	none	提示性消息

(3) POST 用户注册

POST /UserRegister

用户注册接口主要实现用户的注册功能，从前端得到用户id，用户名，用户密码数据，将其放入后端的用户表中，其中用户名和密码不能为空

Body 请求参数

```
{
  "userid": "string",
  "username": "string",
  "passwd": "string"
}
```

请求参数

名称	位置	类型	必选	说明
body	body	object	否	none
» userid	body	string	是	用户id
» username	body	string	是	用户名
» passwd	body	string	是	用户密码

返回示例

成功

```
{
  "ret": "0",
  "msg": "注册成功"
}
```

失败

```
{
  "ret": 1,
  "msg": "用户名或密码为空"
}
```

返回结果

状态码	状态码含义	说明	数据模型
200	OK	成功	Inline
201	Created	失败	Inline

返回数据结构

状态码 200

名称	类型	必选	约束	说明
» ret	string	true	none	0代表成功， 1代表失败
» msg	string	true	none	提示性消息

状态码 201

名称	类型	必选	约束	说明
» ret	integer	true	none	0代表成功， 1代表失败
» msg	string	true	none	提示性消息

(4) GET 获取用户详情

GET / GetUser/{userid}

获取用户详情接口，主要用于实现密码找回功能，通过唯一的userid，来返回密码passwd值。当前端传来的userid是存在的，则正常返回密码，并提示返回成功；若userid不存在，则提示用户不存在的信息

请求参数

名称	位置	类型	必选	说明
userid	path	string	是	用户id

返回示例

成功

```
{
  "ret": 0,
  "passwd": "111",
  "msg": "找回密码成功"
}
```

失败

```
{  
    "ret": 1,  
    "msg": "用户不存在, 请重新输入"  
}
```

返回结果

状态码	状态码含义	说明	数据模型
200	OK	成功	Inline
201	Created	失败	Inline

返回数据结构

状态码 200

名称	类型	必选	约束	说明
» ret	integer	true	none	0代表成功, 1代表失败
» passwd	string	true	none	用户密码
» msg	string	true	none	提示性消息

状态码 201

名称	类型	必选	约束	说明
» ret	integer	true	none	0代表成功, 1代表失败
» msg	string	true	none	提示性消息

2. 日记模块

(1) POST 新建日记

POST /NewDiary

新建日记接口，主要是从前端拿到日记id，日记标题，日记内容，日记修改时间等数据，将其保存至后端的日记表。判断的要求是标题和内容是否为空，若为空，则提示错误信息

Body 请求参数

```
{  
    "diaryid": "string",  
    "title": "string",  
    "content": "string",  
    "time": "string"  
}
```

请求参数

名称	位置	类型	必选	说明
body	body	object	否	none
» diaryid	body	string	是	日记id
» title	body	string	是	日记标题
» content	body	string	是	日记内容
» time	body	string	是	日记修改时间

返回示例

成功

```
{  
    "ret": 0,  
    "msg": "创建日记成功"  
}
```

失败

```
{  
    "ret": 1,  
    "msg": "日记标题和内容不能为空"  
}
```

返回结果

状态码	状态码含义	说明	数据模型
200	OK	成功	Inline
201	Created	失败	Inline

返回数据结构

状态码 200

名称	类型	必选	约束	说明
» ret	integer	true	none	0代表成功，1代表失败
» msg	string	true	none	提示性消息

状态码 201

名称	类型	必选	约束	说明
» ret	integer	true	none	0代表成功，1代表失败
» msg	string	true	none	提示性消息

(2) POST 日记上锁

POST /DiaryLock

日记上锁通过前端得到上锁id和上锁密码，其中上锁id始终和对应的日记id一致，通过上锁id为密码表创建一组新数据，通过日记id，将日记表对应日记的是否上锁改为1.其中后端要判断密码是否为空，为空则提示异常

Body 请求参数

```
{
  "lockid": "string",
  "lockpasswd": "string"
}
```

请求参数

名称	位置	类型	必选	说明
body	body	object	否	none
» lockid	body	string	是	上锁id
» lockpasswd	body	string	是	日记密码

返回示例

成功

```
{
  "ret": 0,
  "msg": "密码创建成功"
}
```

失败

```
{
  "ret": 1,
  "msg": "密码不能为空"
}
```

返回结果

状态码	状态码含义	说明	数据模型
200	OK	成功	Inline
201	Created	失败	Inline

返回数据结构

状态码 200

名称	类型	必选	约束	说明
» ret	integer	true	none	0代表成功，1代表失败
» msg	string	true	none	提示性消息

状态码 201

名称	类型	必选	约束	说明
» ret	integer	true	none	0代表成功，1代表失败
» msg	string	true	none	提示性消息

(3) DELETE 删除日记

DELETE /DeleteDiary

删除日记接口，通过从前端得到日记id参数，来将对应日记表的对应日记删除

Body 请求参数

```
{  
  "diaryid": "string"  
}
```

请求参数

名称	位置	类型	必选	说明
body	body	object	否	none
» diaryid	body	string	是	日记id

返回示例

成功

```
{  
  "ret": 0,  
  "msg": "日记删除成功"  
}
```

返回结果

状态码	状态码含义	说明	数据模型
200	OK	成功	Inline

返回数据结构

状态码 200

名称	类型	必选	约束	说明
» ret	integer	true	none	0代表成功，1代表失败
» msg	string	true	none	提示性消息

(4) GET 获取日记详情

GET /GetDiary/{diaryid}

获取用户详情接口主要是实现点击前端日记时，可以将数据传输至前端，通过从前端得到日记id，来返回日记的各个参数

请求参数

名称	位置	类型	必选	说明
diaryid	path	string	是	日记id

返回示例

成功

```
{  
    "ret": 0,  
    "title": "这是标题",  
    "content": "这是内容",  
    "ifremind": 0,  
    "iflock": 0,  
    "ifpublic": 0  
}
```

返回结果

状态码	状态码含义	说明	数据模型
200	OK	成功	Inline

返回数据结构

状态码 200

名称	类型	必选	约束	说明
» ret	integer	true	none	0代表成功，1代表失败
» title	string	true	none	日记标题
» content	string	true	none	日记内容
» ifremind	integer	true	none	是否设置了日记提醒
» iflock	integer	true	none	是否设置了日记密码
» ifpublic	integer	true	none	日记是否公开

(5) DELETE 取消日记提醒

DELETE /DeleteDiaryRemind

取消日记提醒接口，通过从前端得到提醒id，来将对应的提醒表中的数据删除，并将日记表中对应日记的是否设置提醒值设为0

Body 请求参数

```
{
  "remindid": "string"
}
```

请求参数

名称	位置	类型	必选	说明
body	body	object	否	none
» remindid	body	string	是	提醒id

返回示例

成功

```
{
  "ret": 0,
  "msg": "日记提醒已取消"
}
```

返回结果

状态码	状态码含义	说明	数据模型
200	OK	成功	Inline

返回数据结构

状态码 200

名称	类型	必选	约束	说明
» ret	integer	true	none	0代表成功，1代表失败
» msg	string	true	none	提示性消息

(6) POST 日记解锁

POST /LockCancel

日记解锁用来取消日记的密码锁，从前端得到上锁id和日记密码，根据上锁id将密码表对应数据段删除，由于上锁id和日记id始终一致，因此可以通过日记id，将对应的日记的是否上锁值改为0

Body 请求参数

```
{  
  "lockid": "string",  
  "lockpasswd": "string"  
}
```

请求参数

名称	位置	类型	必选	说明
body	body	object	否	none
» lockid	body	string	是	上锁id
» lockpasswd	body	string	是	日记密码

返回示例

成功

```
{  
  "ret": 0,  
  "msg": "密码锁已删除"  
}
```

返回结果

状态码	状态码含义	说明	数据模型
200	OK	成功	Inline

返回数据结构

状态码 200

名称	类型	必选	约束	说明
» ret	integer	true	none	0代表成功, 1代表失败
» msg	string	true	none	提示性消息

(7) GET 日记按日期搜索

GET /DiaryDateSearch/{date1}{date2}

按日期搜索通过得到前端的时间段，以此参数作为过滤条件，将数据库中对应时间段的所有日记id，以data的形式传给前端

请求参数

名称	位置	类型	必选	说明
date1	path	string	是	开始日期
date2	path	string	是	结束日期

返回示例

成功

```
{  
    "ret": 0,  
    "data": "1 2 3 4 5"  
}
```

返回结果

状态码	状态码含义	说明	数据模型
200	OK	成功	Inline

返回数据结构

状态码 200

名称	类型	必选	约束	说明
» ret	integer	true	none	0代表成功, 1代表失败
» data	string	true	none	包含了所有符合条件的日记id

(8) POST 日记提醒

POST /DiaryRemind

日记提醒接口，主要实现为日记设置闹钟的功能，从前端得到对应的闹钟id和闹钟提醒时间，从而在后端的闹钟表中增加一组对应的数据，并将日记表中的是否设置提醒时间值改为1

Body 请求参数

```
{  
  "remindid": "string",  
  "remindtime": "string"  
}
```

请求参数

名称	位置	类型	必选	说明
body	body	object	否	none
» remindid	body	string	是	提醒id
» remindtime	body	string	是	提醒时间

返回示例

成功

```
{  
  "ret": 0,  
  "msg": "设置成功"  
}
```

返回结果

状态码	状态码含义	说明	数据模型
200	OK	成功	Inline

返回数据结构

状态码 200

名称	类型	必选	约束	说明
» ret	integer	true	none	0代表成功，1代表失败
» msg	string	true	none	提示性消息

(9) GET 日记按关键字搜索

GET /DiarySearch/{content}

按日期搜索通过得到前端的内容，以此参数作为过滤条件，查找日记表，将所有包含此关键字的日记id，以data的形式传给前端

请求参数

名称	位置	类型	必选	说明
content	path	string	是	搜索内容

返回示例

成功

```
{  
    "ret": 0,  
    "data": "1 2 3 4 5"  
}
```

返回结果

状态码	状态码含义	说明	数据模型
200	OK	成功	Inline

返回数据结构

状态码 200

名称	类型	必选	约束	说明
» ret	integer	true	none	0代表成功，1代表失败
» data	string	true	none	包含了所有符合条件的日记id

3.热门日记圈子

(1) POST 关注用户

POST /PersonFollow

关注用户接口，通过从前端得到参数：关注人的个人id和被关注者的id，以及关注发生的时间，从而在关注表中建立数据，并将对应的个人资料表中的关注数和粉丝数进行调整

Body 请求参数

```
{  
    "personid": "string",  
    "personid2": "string",  
    "followtime": "string"  
}
```

请求参数

名称	位置	类型	必选	说明
body	body	object	否	none
» personid	body	string	是	关注人id (主动)
» personid2	body	string	是	被关注人id (被动)
» followtime	body	string	是	关注时间 (应为时间格式, 在apiifox里没有找到)

返回示例

成功

```
{  
    "ret": 0,  
    "msg": "关注成功"  
}
```

返回结果

状态码	状态码含义	说明	数据模型
200	OK	成功	Inline

返回数据结构

状态码 200

名称	类型	必选	约束	说明
» ret	integer	true	none	0代表成功, 1代表失败
» msg	string	true	none	提示消息

(2) DELETE 删除日记评论

DELETE /DeleteDiaryComment

取消日记评论接口, 通过从前端得到评论人id和点赞的日记id, 以及评论的时间, 来确定评论内容是哪一条, 来删除评论表的对应内容, 并将个人详情表中对应用户的评论数进行调整

Body 请求参数

```
{  
    "personid1": "string",  
    "diaryid": "string",  
    "commenttime": "string"  
}
```

请求参数

名称	位置	类型	必选	说明
body	body	object	否	none
» personid1	body	string	是	评论人id
» diaryid	body	string	是	日记id
» commenttime	body	string	是	日记评论时间

返回示例

成功

```
{  
    "ret": 0,  
    "msg": "删除成功"  
}
```

返回结果

状态码	状态码含义	说明	数据模型
200	OK	成功	Inline

返回数据结构

状态码 200

名称	类型	必选	约束	说明
» ret	integer	true	none	0代表成功，1代表失败
» msg	string	true	none	提示性消息

(3) POST 日记点赞

POST /DiaryLike

日记点赞接口，通过得到点赞人和被点赞人的id，以及点赞的日记id、点赞时间，将相关数据写入点赞表中，并将对应的个人详情表中的点赞数进行调整

Body 请求参数

```
{  
    "personid1": "string",  
    "personid2": "string",  
    "diaryid": "string",  
    "liketime": "string"  
}
```

请求参数

名称	位置	类型	必选	说明
body	body	object	否	none
» personid1	body	string	是	点赞人id (主动)
» personid2	body	string	是	被点赞人id (被动)
» diaryid	body	string	是	日记id
» liketime	body	string	是	点赞时间

返回示例

成功

```
{  
    "ret": 0,  
    "msg": "点赞成功"  
}
```

返回结果

状态码	状态码含义	说明	数据模型
200	OK	成功	Inline

返回数据结构

状态码 200

名称	类型	必选	约束	说明
» ret	integer	true	none	0代表成功, 1代表失败
» msg	string	true	none	提示性消息

(4) DELETE 取消日记点赞

DELETE /CancelDiaryLike

取消日记点赞接口，通过从前端得到点赞人id和点赞的日记id，来删除点赞表的对应内容，并将个人详情表中对应用户的点赞数进行调整

Body 请求参数

```
{  
    "personid1": "string",  
    "diaryid": "string"  
}
```

请求参数

名称	位置	类型	必选	说明
body	body	object	否	none
» personid1	body	string	是	点赞人id (主动)
» diaryid	body	string	是	日记id

返回示例

成功

```
{  
    "ret": 0,  
    "msg": "取消点赞成功"  
}
```

返回结果

状态码	状态码含义	说明	数据模型
200	OK	成功	Inline

返回数据结构

状态码 200

名称	类型	必选	约束	说明
» ret	integer	true	none	0代表成功, 1代表失败
» msg	string	true	none	提示性消息

(5) GET 查看热门日记详情

GET /GetPublicDiary/{diaryid}

查看热门日记详情，是通过前端得到日记id，先在日记表中找到对应的字段，将日记的标题等内容传输至前端，再根据日记id从评论表中得到评论内容

请求参数

名称	位置	类型	必选	说明
diaryid	path	string	是	日记id

返回示例

成功

```
{  
    "ret": 0  
}
```

返回结果

状态码	状态码含义	说明	数据模型
200	OK	成功	Inline

返回数据结构

状态码 **200**

名称	类型	必选	约束	说明
» ret	integer	true	none	0代表成功， 1代表失败

(6) DELETE 取消关注用户

DELETE /DeletePersonFollow

取消关注用户接口，通过从前端得到关注者和被关注者的个人id，来删除关注表的对应内容，并将个人详情表中对应用户的粉丝数和关注数进行调整

Body 请求参数

```
{  
    "personid1": "string",  
    "personid2": "string"  
}
```

请求参数

名称	位置	类型	必选	说明
body	body	object	否	none
» personid1	body	string	是	关注人id (主动)
» personid2	body	string	是	被关注人id (被动)

返回示例

成功

```
{  
    "ret": 0,  
    "msg": "取消关注成功"  
}
```

返回结果

状态码	状态码含义	说明	数据模型
200	OK	成功	Inline

返回数据结构

状态码 **200**

0代表成功，1代表失败

名称	类型	必选	约束	说明
» ret	integer	true	none	0代表成功，1代表失败
» msg	string	true	none	提示消息

(7) POST 日记评论

POST /DiaryComment

日记点赞接口，通过得到评论人和被评论人的id，以及评论的日记id、评论内容、评论时间，将相关数据写入评论表中，并将对应的个人详情表中的评论数进行调整

Body 请求参数

```
{  
    "personid1": "string",  
    "personid2": "string",  
    "diaryid": "string",  
    "commentcontent": "string",  
    "commenttime": "string"  
}
```

请求参数

名称	位置	类型	必选	说明
body	body	object	否	none
» personid1	body	string	是	评论人id (主动)
» personid2	body	string	是	被评论人id (被动)
» diaryid	body	string	是	none
» commentcontent	body	string	是	none
» commenttime	body	string	是	none

返回示例

成功

```
{  
    "ret": 0,  
    "msg": "评论成功"  
}
```

返回结果

状态码	状态码含义	说明	数据模型
200	OK	成功	Inline

返回数据结构

状态码 200

名称	类型	必选	约束	说明
» ret	integer	true	none	日记点赞接口，通过得到点赞人和被点赞人的id，以及点赞的日记id、点赞时间，将相关数据写入点赞表中，并将对应的个人详情表中的点赞数进行调整
» msg	string	true	none	提示性消息

4.个人页面

(1) GET 查看获赞列表

GET /GetPersonelLike/{personid}

查看获赞列表接口，通过从前端得到的个人id，在点赞表中对点赞人的id进行查询，将相关数据发送至前端

请求参数

名称	位置	类型	必选	说明
personid	path	string	是	none

返回示例

成功

```
{  
    "ret": 0  
}
```

返回结果

状态码	状态码含义	说明	数据模型
200	OK	成功	Inline

返回数据结构

状态码 200

名称	类型	必选	约束	说明
» ret	integer	true	none	none

(2) PUT 修改个人资料

PUT /ChangePersonel

修改个人资料，用于从前端得到个人id，个人页的用户名，个人签名，来对个人详情表的相关字段进行修改

Body 请求参数

```
{  
  "personid": "string",  
  "personname": "string",  
  "personsing": "string"  
}
```

请求参数

名称	位置	类型	必选	说明
body	body	object	否	none
» personid	body	string	是	none
» personname	body	string	是	none
» personsing	body	string	是	none

返回示例

成功

```
{  
  "ret": 0,  
  "msg": "修改"  
}
```

返回结果

状态码	状态码含义	说明	数据模型
200	OK	成功	Inline

返回数据结构

状态码 200

名称	类型	必选	约束	说明
» ret	integer	true	none	none
» msg	string	true	none	none

(3) POST 创建用户详情

POST /NewPersonel

创建用户详情（用于用户第一次登录时，对有关资料的填写），将构建用户详情的所需参数从前端读取至后端，并在对应的个人详情表中保存

Body 请求参数

```
{  
    "personid": "string",  
    "personname": "string",  
    "personsing": "string"  
}
```

请求参数

名称	位置	类型	必选	说明
body	body	object	否	none
» personid	body	string	是	none
» personname	body	string	是	none
» personsing	body	string	是	none

返回示例

成功

```
{  
    "ret": 0,  
    "msg": "创建成功"  
}
```

失败

```
{  
    "ret": 1,  
    "msg": "用户名不能为空"  
}
```

返回结果

状态码	状态码含义	说明	数据模型
200	OK	成功	Inline
201	Created	失败	Inline

返回数据结构

状态码 200

名称	类型	必选	约束	说明
» ret	integer	true	none	none
» msg	string	true	none	none

状态码 201

名称	类型	必选	约束	说明
» ret	integer	true	none	none
» msg	string	true	none	none

(4) GET 查看获评列表

GET /GetPersonelComment/{personid}

查看获评列表接口，通过从前端得到的个人id，在评论表中对评论人的id进行查询，将相关数据发送至前端

请求参数

名称	位置	类型	必选	说明
personid	path	string	是	none

返回示例

成功

```
{  
    "ret": 0  
}
```

返回结果

状态码	状态码含义	说明	数据模型
200	OK	成功	Inline

返回数据结构

状态码 200

名称	类型	必选	约束	说明
» ret	integer	true	none	none

(5) GET 获取个人资料

GET /GetPersonel/{personid}

请求参数

名称	位置	类型	必选	说明
personid	path	string	是	none

返回示例

返回结果

状态码	状态码含义	说明	数据模型
200	OK	成功	Inline

返回数据结构

五、数据库设计 [陈宇航]

1. 系统数据分析

从用户模块出发，这一块为避免重复注册，应设计一个主键id，同时设计用户名和密码，用于登录、找回密码等操作；

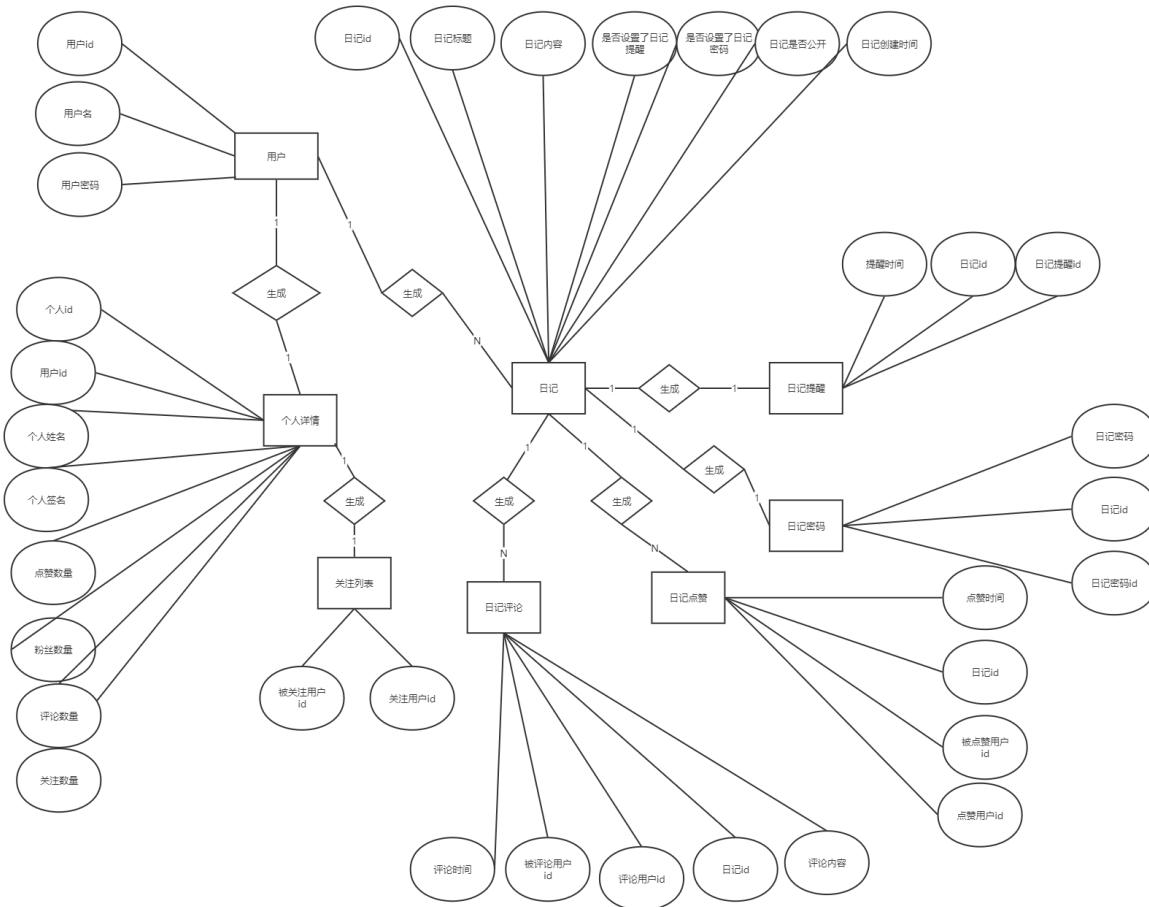
在日记模块方面，日记本身主要由标题、内容、创建时间构成，但还包括了日记是否上锁（若上锁密码是多少），日记是否设置了提醒（若设置了提醒时间是多少），日记是否公开（公开的评论列表和点赞列表等信息）等信息，但这类信息全都放在日记表中太过冗余，因此我选择建立上锁表、提醒表、评论表、点赞表、关注表，将对应的信息通过id串联起来，例如上锁表中的主键上锁id和日记id始终保持一致，双方有一处变化了，另一处也会变化。

最后的个人详情模块，通过用户表中的用户id和个人id相关联，来确保一个用户id对应一个个人id。

综上数据分析，再结合所使用的后端框架django，我决定使用的数据库技术是SQLite+ORM，用软件本身自带的数据库，优点是便于合作时可以直接运行，减少配置环境的麻烦。

2.数据表设计

(1) 整体ER图



(2) 用户表

用户表中的数据主要是包括用户id、用户名、用户密码，将用户id设为主键来作为识别的标识。

名称	类型	必填	说明
» userid	string	是	用户id(主键)
» username	string	是	用户名
» passwd	string	是	用户密码

(3) 日记表

日记表包含的数据类型较多，在一次创建时基本的是需要标题和内容，但考虑到还有诸如公开日记、日记上锁、日记提醒等功能，全都放在一张日记表里非常复杂，因此我选择在日记表只用一个在0和1跳动的值，来代表是否设置了这些功能，从而再通过id一一对应，来找到那些表对应的字段内容。

名称	类型	必填	说明
» diaryid	integer	是	日记id (主键)
» title	string	是	日记标题
» content	string	是	日记内容
» ifremind	integer	否	是否设置了日记提醒 0代表没有设置提醒, 1代表设置了提醒
» iflock	integer	否	是否设置了日记密码 0代表没有设置日记密码, 1代表设置了日记密码
» ifpublic	integer	否	日记是否公开 0代表不公开, 1代表公开
time	date	是	日记创建时间

(4) 上锁表 (日记密码表)

上锁表中的lockid和日记表中的diaryid始终对应保持一致，这样可以保证在对应的日记页面里，能够对日记进行正常的上锁、解锁功能，同时根据一致的id，对两张表的内容进行更改。

名称	类型	必填	说明
» lockid	integer	是	上锁id (主键) 与日记id始终保持一致
» lockpasswd	string	是	日记密码

(5) 日记提醒表 (日记闹钟表)

日记提醒表中的remindid和日记表中的diaryid始终对应保持一致，这样可以保证在对应的日记页面里，能够对日记进行设置日记提醒、取消日记提醒功能，同时根据一致的id，对两张表的内容进行更改。

名称	类型	必填	说明
» remindid	integer	是	提醒id 与日记id始终保持一致
» remindtime	string	是	提醒时间

(6)日记评论表

日记评论表主要是针对选择公开的日记，当一个用户评论另一个用户的日记时，记录两个用户的个人id和日记id，记录至日记评论表，其中personid1是主动评论的那个用户的id，personid2是被评论的那个用户的id，这样可以根据两个用户的id，以及主动和被动的关系，来对对应的个人详情表中的关注数和粉丝数进行操作。同时评论内容和评论删除字段，用于在进入热门日记详情中时展示出来。

名称	类型	必填	说明
» personid1	integer	是	评论人id (主动)
» personid2	integer	是	被评论人id (被动)
» diaryid	integer	是	日记id
» commentcontent	string	是	日记评论内容
» commenttime	date	是	日记评论时间

(7) 日记点赞表

日记点赞表主要是针对选择公开的日记，当一个用户点赞另一个用户的日记时，记录两个用户的个人id和日记id，记录至日记点赞表，其中personid1是主动点赞的那个用户的id，根据该id在个人详情表中对相关用户进行点赞数的调整。

名称	类型	必填	说明
» personid1	integer	是	点赞人id (主动)
» personid2	integer	是	被点赞人id (被动)
» diaryid	string	是	日记id
» liketime	date	是	点赞时间

(8) 关注列表

关注列表记录两个用户之间的关系，来将对应的用户id的对应表单进行调整

名称	位置	类型	必填	说明
» personid1	body	integer	是	关注人id (主动)
» personid2	body	integer	是	被关注人id (被动)

(9)个人详情表

个人详情表主要包括了个人id、个人姓名、个性签名、点赞数量、关注数量、粉丝数量、评论数量字段，其中个人id和用户id始终保持一致，来保证一个用户id对应一个个人id。

名称	类型	必填	说明
personid	integer	是	个人id (主键) 和用户id保持一致
personname	string	是	个人姓名
personsing	string	是	个性签名
likenum	integer	否	点赞数量
follownum	integer	否	关注数量
fansnum	integer	否	粉丝数量
commentnum	integer	否	评论数量

3.数据库相关技术

由于项目使用的是django后端，因此主要用的数据库技术是ORM

3.1 ORM介绍：

ORM，全称Object Relational Mapping，中文叫做对象关系映射，通过ORM我们可以通过类的方式去操作数据库，而不用再写原生的SQL语句。通过把表映射成类，把行作实例，把字段作为属性，ORM在执行对象操作的时候最终还是会把对应的操作转换为数据库原生语句。

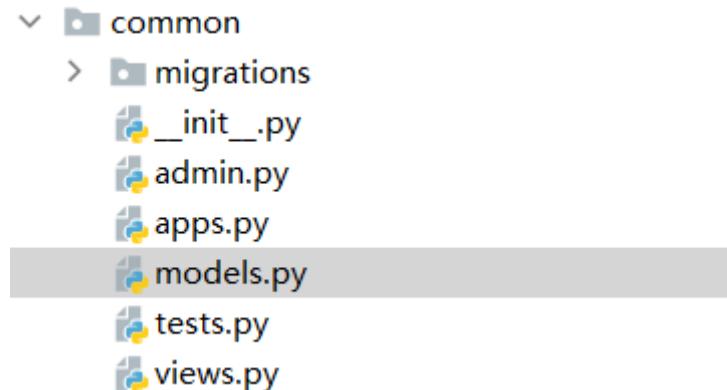
3.2 ORM使用流程：

(1) settings配置数据库

位置在databases字典内，配置项包括引擎，数据库名称，地址，端口号，用户名，密码，这些需要自己写键和键值的

(2) models

在一个APP内部的models内创建模型



```
# 用户表
class User(models.Model):
    # 用户id
    userid = models.CharField(max_length=200, primary_key=True)
    # 用户名称
    username = models.CharField(max_length=30)
    # 用户密码
    passwd = models.CharField(max_length=50)
```

(3) 注册APP

在settings内注册APP, 在INSTALLED_APPS内注册 (添加)

```
from django.contrib import admin

# Register your models here.
from django.contrib import admin

from .models import User,Diary,DiaryLockData,DiaryRemindData,Personel,LikeList,CommentList,FollowList

admin.site.register(User)
admin.site.register(Diary)
admin.site.register(DiaryLockData)
admin.site.register(DiaryRemindData)
admin.site.register(Personel)
admin.site.register(LikeList)
admin.site.register(CommentList)
admin.site.register(FollowList)
```

(4)生成迁移文件

python manage.py makemigrations

(5) 迁移数据库

python manage.py migrate

(6)可以在ORM中看到数据库了

Django 管理

站点管理

COMMON		
Comment lists		
Diary lock datas		
Diary remind datas		
Diaries		
Follow lists		
Like lists		
Personels		
Users		

(7)在views内定义相关的增删改查操作

修改 user

User object (1)

历史

UserId:	<input type="text" value="1"/>
Username:	<input type="text" value="cyh"/>
Password:	<input type="text" value="123"/>

六、AndroidStudio前端的实现 [陈宇航、李浩腾]

前端自定工具类：HttpUtil

用于实现前后端数据的通信，为四类接口定义方法，实现便利的调用（此处已对数据的传输进行了说明，因此在下文向后端传送数据的具体实现中不多赘述）

1.GET类型接口

```
//callback是okhttp自带的回调接口，这里写的是使用GET方式获取服务器数据
fun sendOkHttpRequest(address: String, callback: Callback) {
    val client = OkHttpClient()
    val request = Request.Builder()
        .url(address)
        .build()
    //enqueue方法内部已经帮助我们开启好了线程，最终的结果会回调到callback中
    client.newCall(request).enqueue(callback)
}
```

2.POST类型接口

```
//使用POST方式向服务器提交数据并获取返回提示数据
fun sendOkHttpResponse(
    address: String,
    requestBody: RequestBody, callback: callback
) {
    val client = OkHttpClient()
    //JSONObject这里是要提交的数据部分
    val request: Request = Request.Builder()
        .url(address)
        .post(requestBody)
        .build()
    client.newCall(request).enqueue(callback)
}
```

3.PUT类型接口

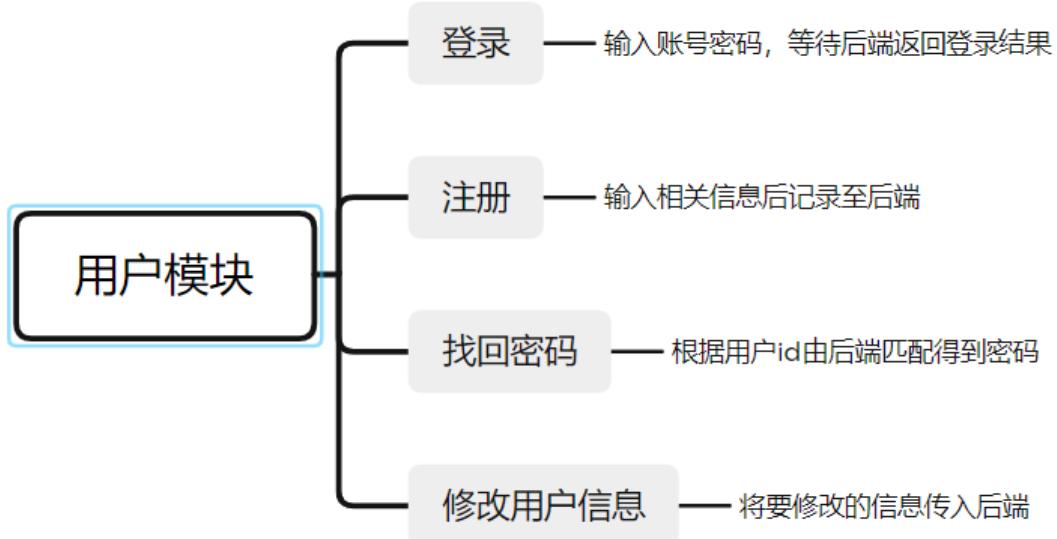
```
//使用POST方式向服务器提交数据并获取返回提示数据
fun sendOkHttpResponse(
    address: String,
    requestBody: RequestBody, callback: callback
) {
    val client = OkHttpClient()
    //JSONObject这里是要提交的数据部分
    val request: Request = Request.Builder()
        .url(address)
        .post(requestBody)
        .build()
    client.newCall(request).enqueue(callback)
}
```

4.DELETE类型接口

```
//使用DELETE方式向服务器提交数据并获取返回提示数据
fun sendOkHttpDelete(
    address: String, callback: callback
) {
    val client = OkHttpClient()
    //JSONObject这里是要提交的数据部分
    val request: Request = Request.Builder()
        .url(address)
        .delete()
        .build()
    client.newCall(request).enqueue(callback)
}
```

6.1 用户登录模块实现[陈宇航]

模块功能总览



用户模块整体较为常规，实现方法也很简单，这里简要阐述下

1. 用户登录

用户登录即通过读取两个TextView的值，得到用户名和密码，将其和数据库进行匹配，若成功则跳转到主页面，若匹配不成功则提示错误信息。



2. 用户注册

用户注册功能主要是把用户名和密码传输至服务端，并保存至数据库，这里在前端主要负责判断两次密码是否一致，若不一致提示重新输入，同时采用双保险的思路，前端也对密码和用户名是否为空进行一次判断。正常注册成功后，自动跳转回用户登录页面，并通过Intent将用户名显示在用户名输入框，来方便用户使用

请输入用户名

请输入密码

请再次输入密码

注 册

3. 用户找回密码

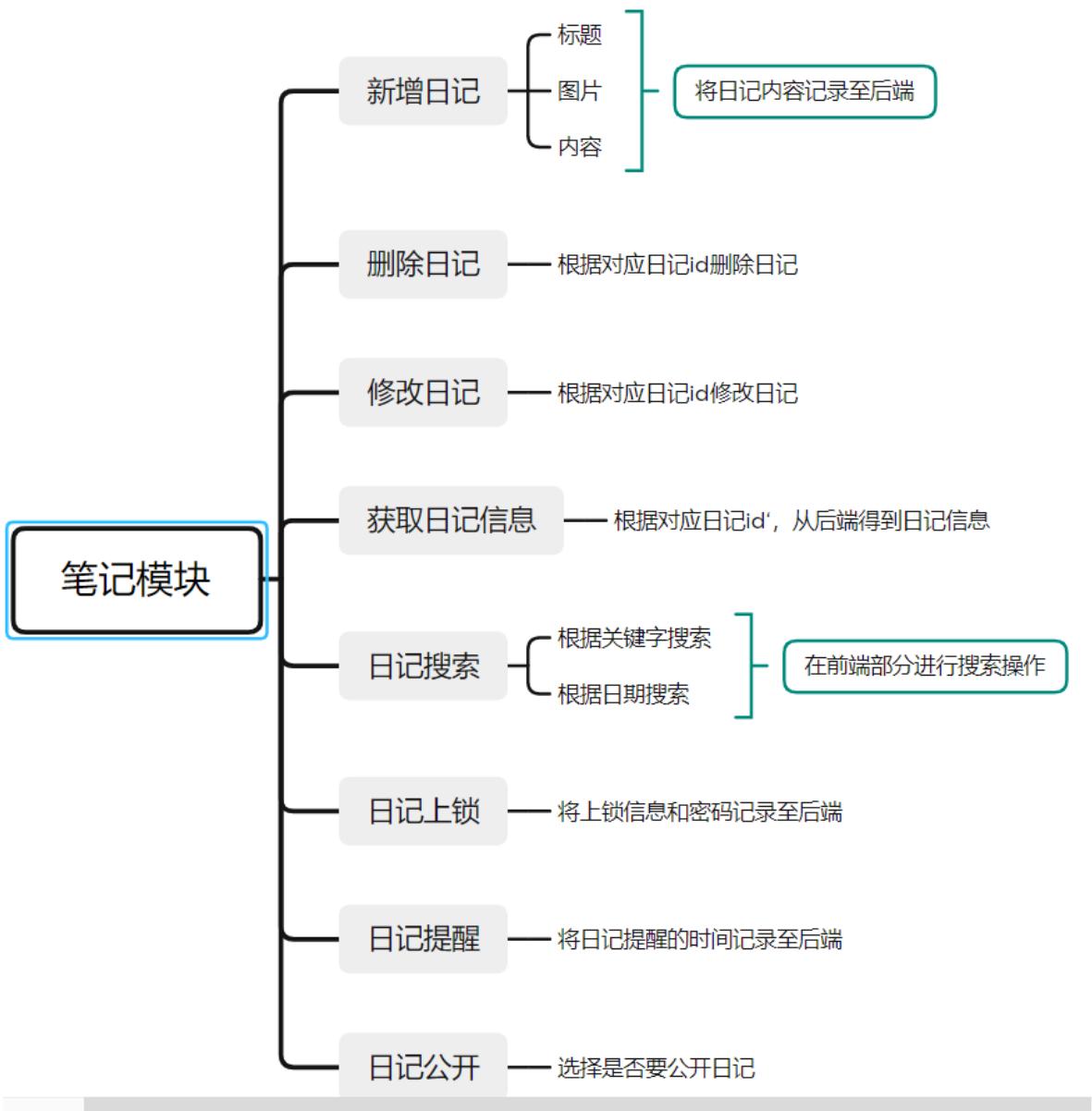
用户找回密码主要的依据是用户id，因此在数据库中用户id是主键，可以确定唯一关系，这一部分设计的较为简单

请输入要找回的用户名

确定找回

6.2 日记模块实现[陈宇航]

模块功能总览



1.新建日记

新增日记功能主要是将日记的标题、内容等信息传输至后端，并在后端数据库中进行相关字段的保存，主要的难点在于数据的转换，如何整理前端的数据，并将其传入至服务端。这里的传输利用的是POST接口。

前端技术方面，使用getTitle方法（具体在展示在代码中），截取标题，作为之后主页面的预览

```
// 截取EditText中的前一部分作为标题，用于显示在主页列表中
title = getTitle(context);
```

使用Date函数来获取当前时间，记录日记创建时间传入后端

```
Date curDate = new Date(System.currentTimeMillis());// 获取当前时间
time = formatter.format(curDate);
```

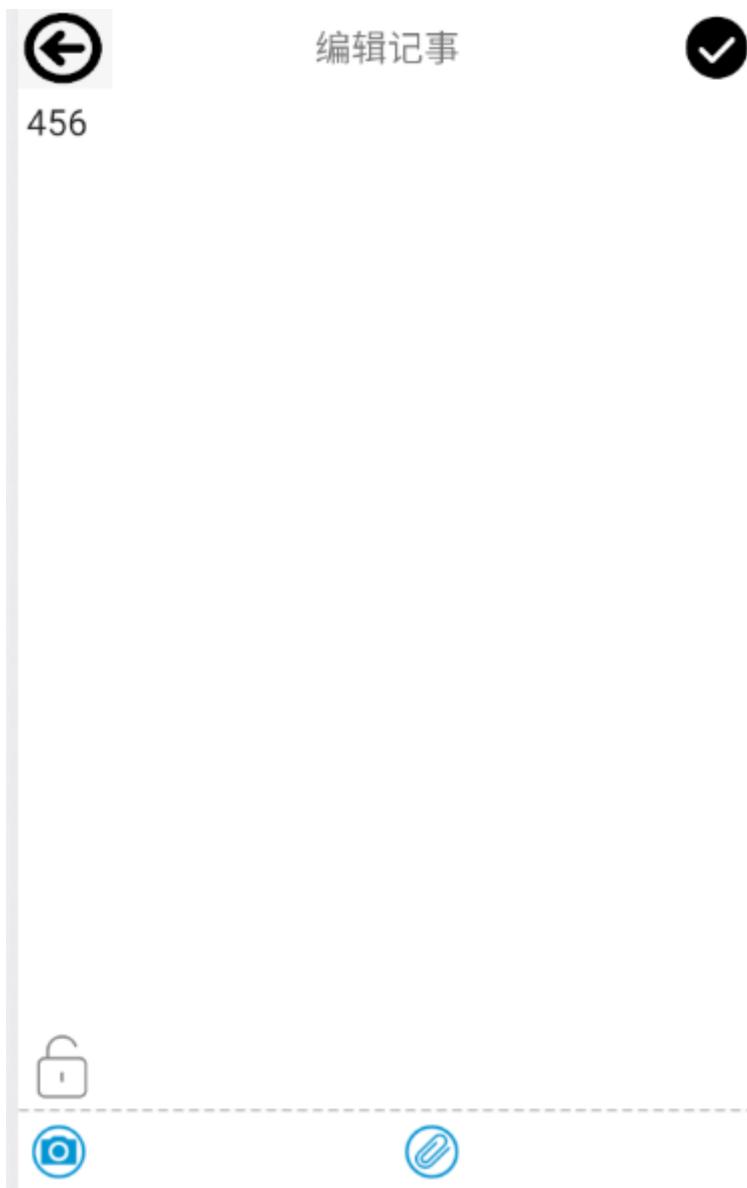
利用菜单栏适配器，将上锁、拍照、提醒等功能集中在底部的菜单栏

```
//配置菜单方法
private void initBottomMenu() {
    //菜单集合
    ArrayList<Map<String, Object>> menus = new ArrayList<Map<String, Object>>();
```

```
for (int i = 0; i < bottomItems.length; i++) {//循环菜单集合
    Map<String, Object> item = new HashMap<String, Object>();
    item.put("image", bottomItems[i]); //循环图片集合添加到菜单中
    menus.add(item); //添加图片菜单到底部菜单
}
//菜单长度
bottomMenu.setNumColumns(bottomItems.length);
//底部菜单
bottomMenu.setSelector(R.drawable.bottom_item);
//实例化底部菜单适配器
SimpleAdapter mAdapter = new SimpleAdapter(Addnote.this, menus,
        R.layout.item_button, new String[]{"image"},
        new int[]{R.id.item_image});
bottomMenu.setAdapter(mAdapter); //为底部菜单添加适配器
}
```

日记新建成功后，在主页的保存是通过新建的适配器来完成的，使用新建的AdaptView和AdaptScrollView，来使日记能够美观地显示在主页

整体效果图如下：



(新建完成后的主页预览)



2.删除日记

删除日记主要是在主界面完成，使用ItemLongClickListener监听器，长按一个日记后跳出选择窗口，该窗口为实例化AlertDialog后进行的添加，主要有删除和编辑两个选项，使用switch选择函数，来响应控件的onClick事件。

点击编辑后，此时根据通过获取日记详情接口，通过判断数据库中的是否上锁值，即日记有无密码进行判断，若有密码，则必须输入密码以后才可以删除，若无密码，可以直接进行删除。

如下图展示效果：



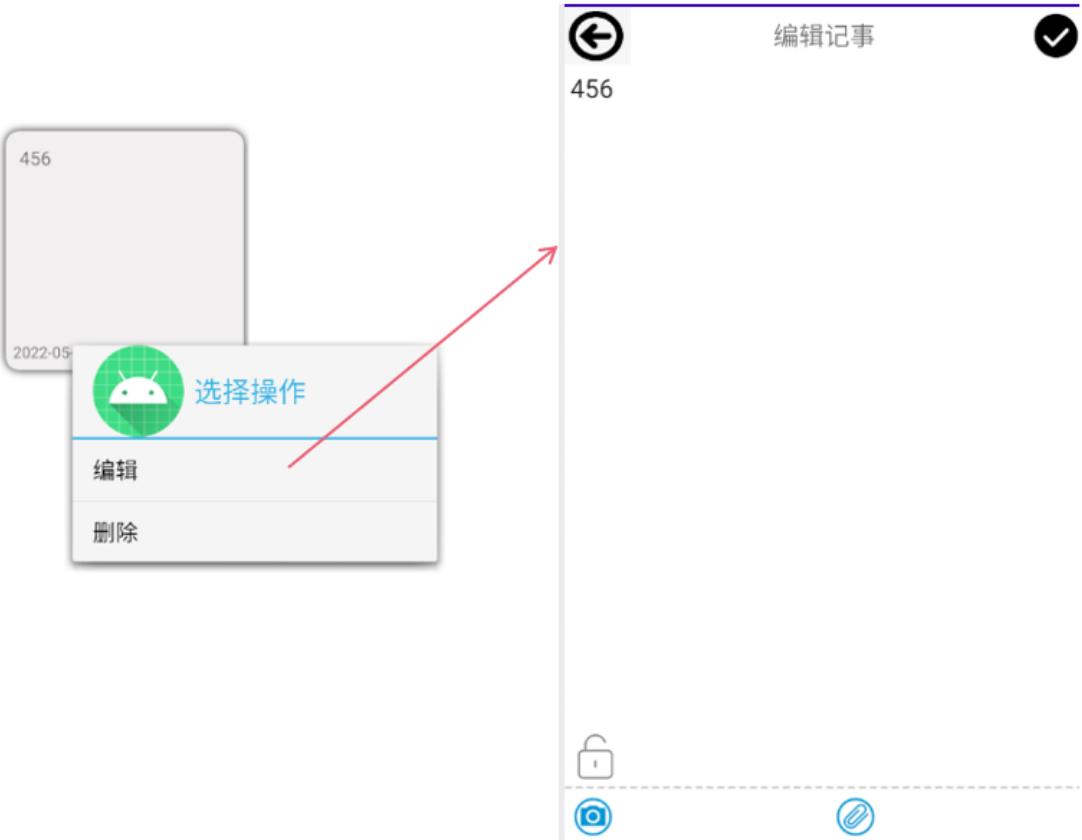
(删除后)



3.修改日记

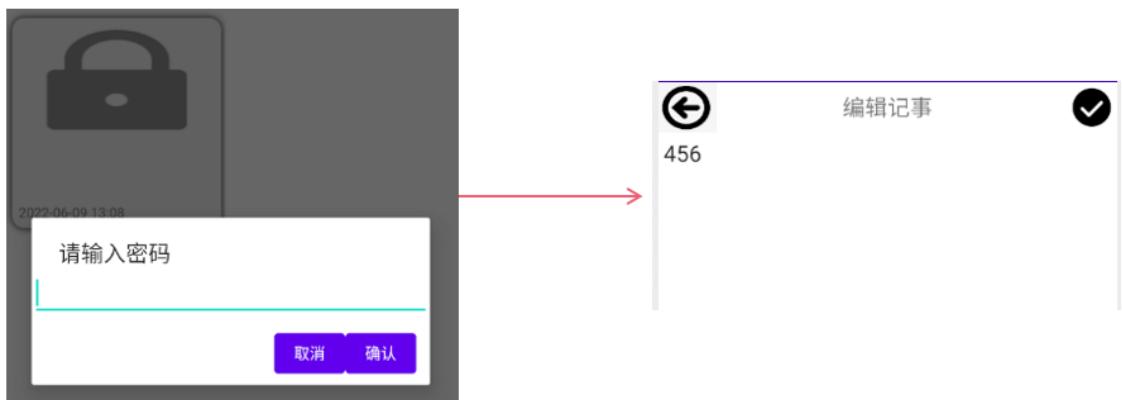
修改日在前端的界面操作和删除日记类似，也是在主界面完成，使用ItemLongClickListener监听器，长按一个日记后跳出选择窗口，该窗口为实例化AlertDialog后进行的添加，主要有删除和编辑两个选项，使用switch选择函数，来响应控件的onClick事件。

选择编辑按钮后，跳转至日记页面，将日记得有关信息修改后，传输至服务端



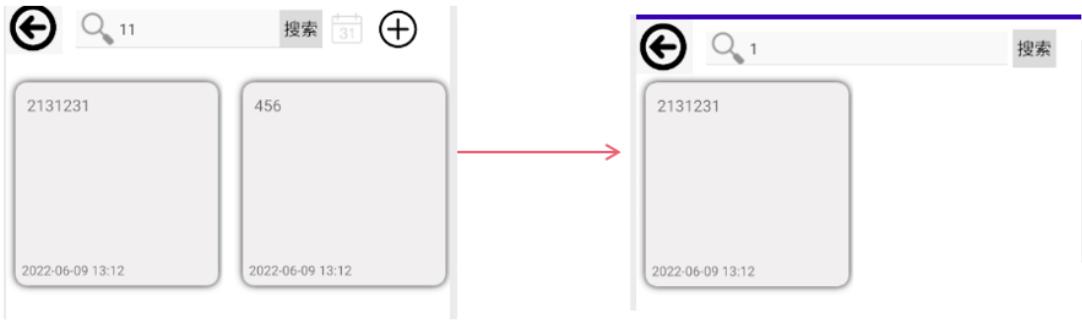
4.获取日记信息

获取日记信息主要是在点击日记时，得到日记的id，从后端根据唯一的id得到日记的相关数据，并将其渲染至前端页面，若日记有密码，则需要输入正确的密码后进入。



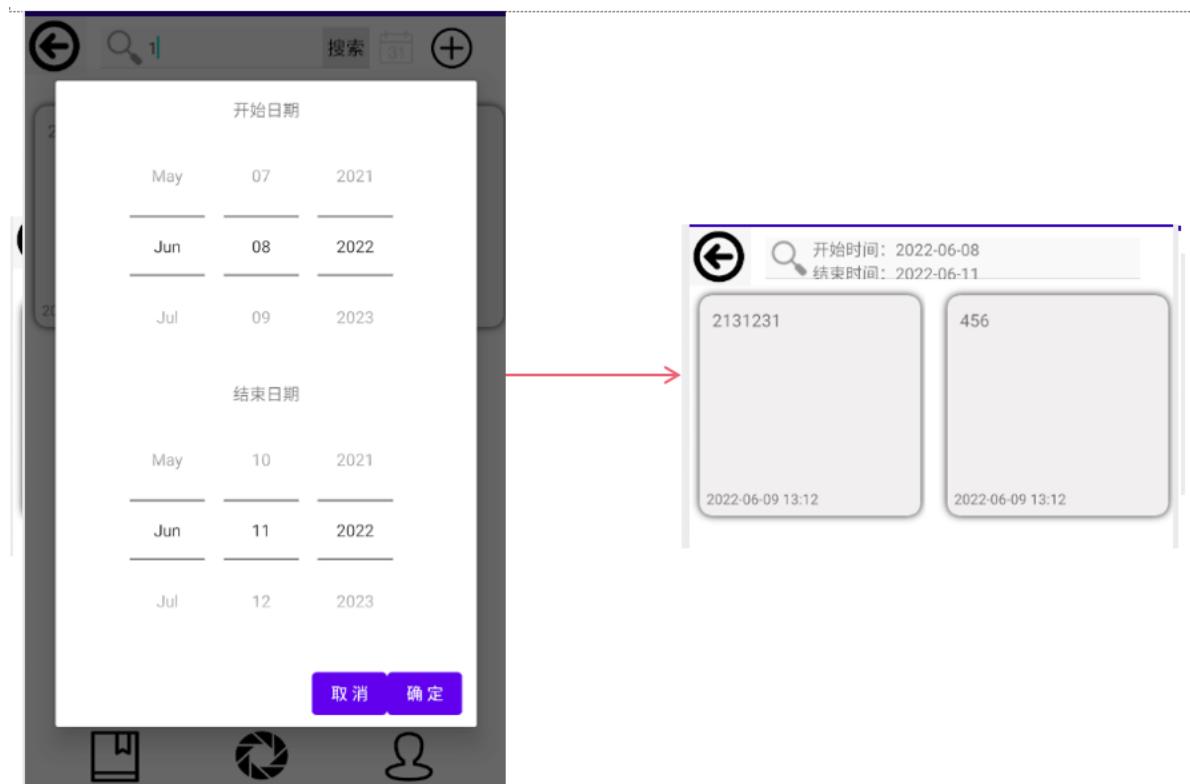
5.日记按关键字搜索

通过获取搜索文本框上的控件值，将内容作为content参数传输至后端，后端在数据库中进行过滤，把过滤后的日记id传输至前端，前端根据对应的日记id，查询对应的日记详情，并使用AdaptView和AdaptScrollView来显示在主页面上



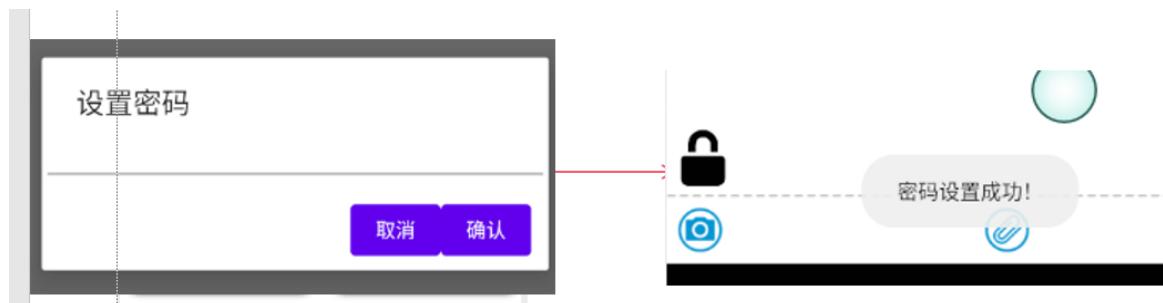
6. 日记按日期搜索

前端日期的选择基于DateTimePicker实现，通过两个，选择好时间范围后，传输至后端，后端根据时间范围，在日记表中搜寻该段时间内所有的日记id，使用id查询对应的日记详情，并使用AdaptView和AdaptScrollView来显示在主页面上



7. 日记上锁

日记上锁主要是在日记的详情页完成，点击底部的上锁按钮，选择密码值，使用if语句进行判空操作，若不为空则传输至后端，待收到后端的设置成功提醒后，前端也生成toast设置成功，并对imageView上锁控件的图片进行变换，改为上锁状态。



8.设置日期提醒

日期提醒主要是使用了AndroidStudio的Alarm技术，通过获取时间，来保存至数据库，当每一次进入系统时就会进行判断，若时间一致，就会进行提醒



6.3热门日记圈子模块[李浩腾]

模块功能总览

笔记圈子模块主要用于分享每个人的日记功能，可以把大家的经历一起分享，评论，点赞，收藏。主要需要实现功能如下：

1.获取用户详情

获取用户详情首先通过创建User用户类分别设定，id用于数据库检索，账户，密码，昵称，头像，关注，粉丝，获赞，评论的数据。

```
public class User extends LitePalSupport {  
    //id，必须有的，数据库的检索标识  
    private int id;  
    //用户账户  
    private String account;  
    //用户密码  
    private String password;  
    //用户昵称  
    private String nickname;  
    //用户头像，只存储URI，需要时去服务器获取  
    private String headPhoto;  
    //关注列表  
    private String guanzhu;  
    //粉丝列表  
    private String fensi;  
    //获赞数  
    private String huozanshu;  
    //评论数  
    private String pinglun;  
    //下面分别实现相应的get和set方法
```

接着创建一个UserUtils的获取用户工具类

```
//获取用户信息的工具类
public class UserUtils {
    //获取当前登录用户的信息
    public static User getCurrentUser(){
        User user = new User();
        int id = -1;
        //返回登录的用户数据
        List<User> users = LitePal.findAll(User.class);
        for(User u : users){
            if(u.isLogin()){
                id = u.getId();
            }
        }
        user = LitePal.find(User.class, id);
        return user;
    }

}
```

在需要读取后端用户数据时，通过id进行检索，通信依赖于Http协议。我们创建HttpUtils类用于进行发送和接受请求到服务器。大致分为请求和接受两个部分完成。

我们使用params请求体内容，encode编码格式发送POST请求到服务器来获取请求体，同时要先设置请求体的文本类型，长度。在获取输出流时，向服务器写入数据在获得服务器的响应码后，处理服务器的响应结果。同时也需要接收来自服务器的响应结果，将输入流转化为字符串。最后我们将接收的字符串写入到User类中完成获取用户详情的功能。

2. 关注用户

本功能主要用于记录用户关注的其他用户，由于每个用户都有其 id，我们只需要确认当前界面中用户的id，和是否已经关注。由于是否关注在获取本用户信息的时候已经有了记录，主要需要将此页面的用户 id 传送至后端的数据库中。

3. 获取公开日记

获取公开日记功能主要需要从后端服务器中读取三部分的数据：作者用户id，封面，日记标题三个部分。其中作者用户id作为主键发送请求至后端数据库中进行查询，接受返回的输出流，获取后，需要将数据渲染到界面上。

以这样一个小单元为一篇文章的容器，同时我们需要创建一个类用于存放需要的信息。

```
public class NewsBean implements Serializable {

    private String title;
    private String content;
    private int imageResourceId;
```

```
public NewsBean() {  
}  
  
public NewsBean(String title, String content, int imageResourceId) {  
    this.title = title;  
    this.content = content;  
    this.imageResourceId = imageResourceId;  
}
```

同时总的页面我们采用了RecyclerView的瀑布流的布局，将每一个NewsBean放入到定义的mMyAdapter中，一起进行渲染。并且，还需要给留下setOnClickListener的点击事件用于转到日记详情界面。

4. 评论

查看评论功能主要需要从后端读取两项数据，评论人 id，评论的内容，用params的结构从后端读取评论的数据用message的形式输出。

```
Map<String, String> params = new HashMap<String, String>();  
params.put("requesttop", "getcomments");  
params.put("gid", uagdata.get("gid").toString());  
String strUrlpath = getResources().getString(R.string.burl) + "Goodsdetail_Servlet";  
String Result = HttpUtils.submitPostData(strUrlpath, params, "utf-8");  
System.out.println("结果为：" + Result);  
Message message = new Message();  
message.what = 1;  
message.obj = Result;  
handler.sendMessage(message);
```

发表评论与查看评论结构相似，但是需要有更多的传入数据如下：

```
params.put("requesttop", "comments");  
params.put("account", user.getAccount());  
params.put("uid", uagdata.get("uid").toString());  
params.put("gid", uagdata.get("gid").toString());  
params.put("gname", uagdata.get("gname").toString());  
params.put("constate", "1");
```

用于发表评论给别人显示uid和 g i d 分别表示文章的作者和文章的id。将 p a r a m s 传入后端数据库。

5. 点赞

点赞功能与评论功能类似，但是结构简单得多，仅仅需要判断是否已经点赞即可，只需要传入uid和 g i d 进行查询，采用 i n t 型变量 0 - 1 来判断即可。

6.4个人界面模块[李浩腾]

模块功能总览

个人页面模块主要用于展示用户的各种信息，主要分为两个部分，本用户和其他用户。两个界面都可以混用。

1.个人详情

获取个人详情的功能与上一个模块的获取用户详情的功能类似，获取用户详情首先通过创建User类分别设定，id用于数据库检索，账户，密码，昵称，头像，关注，粉丝，获赞，评论的数据。

其中又有三个页面需要进行创建，关注、粉丝、获赞三个界面。在个人页面部分也加入了日记展示的部分，同样采用RecyclerView的瀑布流的布局进行渲染。

三个界面布局结构功能基本相似，都采用Adapter来进行。

```
inner class ViewHolder(view: View) : RecyclerView.ViewHolder(view) {  
    val conImage: ImageView = view.findViewById(R.id.Fans_imageview)  
    val conName: TextView = view.findViewById(R.id.Fans_name)  
    val signature: TextView = view.findViewById(R.id.Fans_signature)  
}
```

仅贴出粉丝页面的大致代码，其余页面逻辑与其基本一致。布局的样式图为

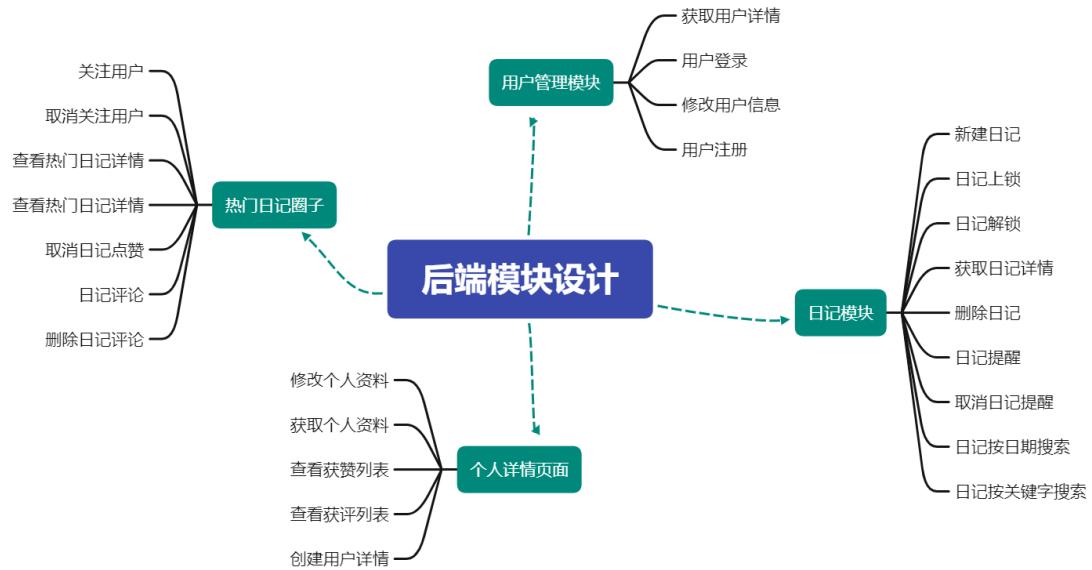
个人界面布局基本如下：

个人公开日记：

需要实现的功能要求从后端获取该用户全部类别为公开的日记，随后可以上一个模块中日记展示圈子页面的格式在主页中进行展示，我们同样采用RecyclerView的瀑布流的布局，将每一个NewsBean放入到定义的mMyAdapter中，一起进行渲染。并且，还需要给留下setOnClickListener的点击事件用于转到日记详情界面。传入后端的数据较为简单，仅仅只需要查询id和是否公开类别的并集即可。

七、django后端的实现[陈宇航]

7.1后端模块总览



7.1 用户管理模块的实现

1. 获取用户详情

获取用户详情主要用于的是找回密码功能，`request.params['userid']`语句从前端得到userid参数，以此参数在User表中进行查询；查询的语句 `User.objects.get(userid=userid)`放入异常判断语句中，若不存在，则向前端返回“用户不存在，请重新输入”的信息，若存在，则用数据查询语句

`User.objects.values("passwd").filter(userid=userid)`在数据表中查找出对应字段的密码，并将其传输至前端，提示找回密码成功。

```
# 获取用户详情(用于找回密码)
def GetUser(request):
    global user
    request.params = json.loads(request.body)
    userid = request.params['userid']
    try:
        user = User.objects.get(userid=userid)
    except User.DoesNotExist:
        return {
            'ret': 1,
            'msg': "用户不存在，请重新输入"
        }

    passwddata=User.objects.values("passwd").filter(userid=userid)
    data = list(passwddata)
    passwd = data[0]['passwd']

    return JsonResponse({
        'ret': 0,
        'msg':"找回密码成功",
        'passwd': passwd
    })
```

2.用户登录

用户登录是形式为get的接口，通过request.GET.get('username')和passwd = request.GET.get('passwd')，来得到用户名和用户密码，用User.objects.values().filter(username=username, passwd=passwd)语句将这两个参数在数据表里进行筛选，若存在则返回登录成功的信息，前端可以进行页面的跳转，若失败则返回登录失败的信息，前端收到信息后，用toast提示用户用户名或密码错误。

```
# 用户登录
def UserLogin(request):
    username = request.GET.get('username')
    passwd = request.GET.get('passwd')
    type = User.objects.values().filter(username=username, passwd=passwd)
    if type:
        return JsonResponse({'ret': 0, 'msg': '登录成功'})
    else:
        return JsonResponse({'ret': 1, 'msg': '登陆失败'})
```

3.修改用户信息

修改用户信息为put接口，从前端得到用户id，用户名，用户密码，由于用户id是主键，因此将用户id作为唯一的标识符，用User.objects.get(userid=userid)对数据库进行过滤，若不存在则向前端返回用户不存在的信息；若存在，则将对应的用户名和密码进行修改，并向前端返回“修改成功”的信息。

```
# 修改用户详情
def userChange(request):
    global UserChange

    request.params = json.loads(request.body)
    userid = request.params['userid']
    username = request.params['username']
    passwd = request.params['passwd']
    try:
        UserChange = User.objects.get(userid=userid)
    except:
        return JsonResponse({'ret': 1, "msg": "用户不存在，请重新输入"})

    if username:
        UserChange.username = username
    if passwd:
        UserChange.passwd = passwd
    UserChange.save()

    return JsonResponse({'ret': 0, 'msg': '修改成功'})
```

4.用户注册

用户注册使用userid = request.params['userid']、username = request.params['username']、passwd = request.params['passwd']语句从前端得到用户id、用户名、用户密码等信息，用if username and passwd判断用户名和密码中是否有空值，若有，则返回“用户名或密码为空”信息给前端，若无，则用User.objects.create(userid=userid, username=username, passwd=passwd)在用户表语句中创建数据信息，并向前端返回“注册成功”的信息。

```

# 用户注册
def UserRegister(request):
    request.params = json.loads(request.body)
    userid = request.params['userid']
    username = request.params['username']
    passwd = request.params['passwd']
    if username and passwd:
        User.objects.create(userid=userid, username=username, passwd=passwd)
        return JsonResponse({'ret': 0, 'msg': '注册成功'})
    else:
        return JsonResponse({'ret': 1, 'msg': '用户名或密码为空'})

```

7.2日记模块的实现

1.新增日记

新增日记功能，使用diaryid = request.params['diaryid']、title = request.params['title']、content = request.params['content']语句来从前端得到日记id、日记标题、日记内容，使用if语句if title and content判断标题和内容中是否为空，若空则向前端返回‘日记标题和内容不能为空’信息，若不为空，则用Diary.objects.create(diaryid=diaryid, title=title, content=content)在日记表中创建相关数据。

```

# 新增日记
def NewDiary(request):
    request.params = json.loads(request.body)
    diaryid = request.params['diaryid']
    title = request.params['title']
    content = request.params['content']

    if title and content:
        Diary.objects.create(diaryid=diaryid, title=title, content=content)
        return JsonResponse({'ret': 0, 'msg': '创建日记成功'})
    else:
        return JsonResponse({'ret': 1, 'msg': '日记标题和内容不能为空'})

```

2.日记上锁

日记上锁功能，通过lockid = request.params['lockid']，lockpasswd = request.params['lockpasswd']接受前端传来的上锁id和密码，若密码为空，则向前端发送“密码不能为空”的消息；若不为空，先用DiaryLockData.objects.create(lockid=lockid, lockpasswd=lockpasswd)语句在上锁表中创建新字段，diarychange = Diary.objects.get(diaryid=diaryid)语句在日记表中找到对应的字段，用diarychange.iflock = 1语句将是否上锁的值改为1

```

# 日记上锁
def DiaryLock(request):
    request.params = json.loads(request.body)
    global diarychange
    # lockid和diaryid应始终保持一致
    lockid = request.params['lockid']
    lockpasswd = request.params['lockpasswd']
    diaryid = lockid

    if lockpasswd:
        DiaryLockData.objects.create(lockid=lockid, lockpasswd=lockpasswd)

```

```

# 将日记表的是否有密码更改为1，即代表有密码
diarychange = Diary.objects.get(diaryid=diaryid)
diarychange.iflock = 1
diarychange.save()
return JsonResponse({'ret': 0, 'msg': '密码创建成功'})
else:
    return JsonResponse({'ret': 1, 'msg': '密码不能为空'})

```

3.日记解锁

日记解锁用request.params['lockid']语句接受前端的参数lockid，根据该参数，在对应的密码数据表中删除对应字段，同时根据对应的日记id，将日记表中对应的是否上锁值改为0。

```

# 日记解锁
def LockCancel(request):
    request.params = json.loads(request.body)
    global diarychange, lockdatachange
    # lockid和diaryid应始终保持一致
    lockid = request.params['lockid']

    # 日记的是否上锁改为0
    diaryid = lockid
    diarychange = Diary.objects.get(diaryid=diaryid)
    diarychange.iflock = 0
    diarychange.save()

    lockdatachange = DiaryLockData.objects.get(lockid=lockid)
    lockdatachange.delete()
    return JsonResponse({'ret': 0, 'msg': '密码锁已删除'})

```

4.获取日记详情

获取日记详情是GET类型接口，使用request.GET.get('diaryid')从前端得到日记id，根据id来使用Diary.objects.values("title").filter(diaryid=diaryid)在数据表中对标题部分进行过滤，list之后得到字典类型的值，取出其标题数据并发送至前端。得到日记正文内容也是同理。

```

# 获取日记详情
def GetDiary(request):
    diaryid=request.GET.get('diaryid')
    #得到日记标题内容
    titledata = Diary.objects.values("title").filter(diaryid=diaryid)
    data1 = list(titledata)
    title = data1[0]['title']
    print(title)
    #得到日记正文内容
    contentdata = Diary.objects.values("content").filter(diaryid=diaryid)
    data2 = list(contentdata)
    content = data2[0]['content']
    print(content)
    return JsonResponse({'ret': 0,
                        'title': title,
                        'content': content
                      })

```

5.日记删除

日记删除对应的是DELETE型接口，request.params['diaryid']语句从前端接受日记id信息，在数据表用日记id进行查询后，适用diarychange.delete()语句进行删除，同时，用if语句判断该日记是否有密码，即判断iflock参数是否为1，若有密码，则对应的密码表也随之删除。

```
# 日记删除
def DeleteDiary(request):
    request.params = json.loads(request.body)
    global diarychange, lockdatachange
    diaryid = request.params['diaryid']
    lockid = diaryid
    # 删除日记
    diarychange = Diary.objects.get(diaryid=diaryid)

    # 如果日记有密码锁，日记删除，密码锁也跟着删除
    print(diarychange.iflock)
    if diarychange.iflock == 1:
        lockdatachange = DiaryLockData.objects.get(lockid=lockid)
        lockdatachange.delete()
    diarychange.delete()
    return JsonResponse({'ret': 0, 'msg': '日记删除成功'})
```

6.创建日记提醒

创建日记提醒，用request.params['remindid']从前端的到提醒id，该id和日记id始终保持一致；request.params['remindtime']得到日记提醒时间，之后使用DiaryRemindData.objects.create(remindid=remindid, remindtime=remindtime)语句进行数据库的创建，同时，要依据日记id，在日记表中找到对应日记，将是否设置提醒参数值设为1.

```
#创建日记提醒
def DiaryRemind(request):
    request.params = json.loads(request.body)
    global diarychange
    # remindid和diaryid应始终保持一致
    remindid = request.params['remindid']
    remindtime = request.params['remindtime']
    diaryid = remindid

    DiaryRemindData.objects.create(remindid=remindid, remindtime=remindtime)
    diarychange = Diary.objects.get(diaryid=diaryid)
    diarychange.ifremind = 1
    diarychange.save()
    return JsonResponse({'ret': 0, 'msg': '设置成功'})
```

7.取消日记提醒时间

创建日记提醒，用request.params['remindid']从前端的到提醒id，该id和日记id始终保持一致，根据该id，在日记提醒表里删除对应字段内容reminddatachange.delete()，同时将日记表中的是否设置提醒时间设为0

```
#取消日记提醒时间
def DeleteDiaryRemind(request):
    request.params = json.loads(request.body)
```

```

global diarychange, reminddatachange
# remindid和diaryid应始终保持一致
remindid = request.params['remindid']

# 日记的是否设置提醒时间为0
diaryid = remindid
diarychange = Diary.objects.get(diaryid=diaryid)
diarychange.ifremind = 0
diarychange.save()

reminddatachange = DiaryRemindData.objects.get(remindid=remindid)
reminddatachange.delete()
return JsonResponse({'ret': 0, 'msg': '日记提醒已取消'})

```

8.日记按内容搜索

日记按内容搜索，request.GET.get('content')从前端得到需要搜索的内容，根据content内容在日记表里进行搜索，搜索的范围是内容是否包含content，返回的结果是所有包含该内容的日记id，将其以data形式传送至前端return JsonResponse({'ret': 'data':data})

```

#日记按内容搜索
def DiarySearch(request):
    content= request.GET.get('content')
    data =
    list(Diary.objects.values("diaryid").filter(content__contains=content))
    return JsonResponse({'ret': 0,
                        'data':data})

```

7.3热门日记圈模块的实现

1.关注用户

关注用户，personid1 = request.params['personid1']从前端接受关注人的id，request.params['personid2']从前端接受被关注人的id，request.params['followtime']从前端接受关注时间。这里的两个个人id之间，有着主动和被动的关系，主动方应在个人详情表中的关注数加1，被动方应在个人详情表中的粉丝数加1。最后在关注表中创建新数据，FollowList.objects.create(personid1=personid1, personid2=personid2, followtime=followtime)，向前端返回关注成功的信息。

```

# 关注用户
def PersonFollow(request):
    request.params = json.loads(request.body)
    #关注人id（主动）
    personid1 = request.params['personid1']
    #被关注人id（被动）
    personid2 = request.params['personid2']
    #关注时间
    followtime=request.params['followtime']

    global Person1Change,Person2Change
    Person1Change = Personnel.objects.get(personid=personid1)
    Person1Change.follownum=Person1Change.follownum+1
    Person1Change.save()

    Person2Change=Personnel.objects.get(personid=personid2)

```

```

Person2Change.fansnum = Person2Change.fansnum + 1
Person2Change.save()

FollowList.objects.create(personid1=personid1, personid2=personid2, followtime=followtime)

return JsonResponse({'ret': 0, 'msg': '关注成功'})

```

2.取消关注

取消关注功能， personid1 = request.params['personid1']从前端接受关注人的id，request.params['personid2']从前端接受被关注人的id，根据这两个id的对应关系，在关注表中删除对应内容， FollowListChange = FollowList.objects.get(personid1=personid1) FollowListChange.delete()，同时，在个人详情表中，主动关注人的关注数减1，被关注人的粉丝数减1。

```

#取消关注
def DeletePersonFollow(request):
    request.params = json.loads(request.body)
    #关注人id（主动）
    personid1 = request.params['personid1']
    #被关注人id（被动）
    personid2 = request.params['personid2']

    global Person1Change, Person2Change, FollowListChange
    Person1Change = Person1.objects.get(personid=personid1)
    Person1Change.follownum=Person1Change.follownum-1
    Person1Change.save()

    Person2Change=Person1.objects.get(personid=personid2)
    Person2Change.fansnum = Person2Change.fansnum-1
    Person2Change.save()

    FollowListChange = FollowList.objects.get(personid1=personid1)
    FollowListChange.delete()

    return JsonResponse({'ret': 0, 'msg': '取消关注成功'})

```

3.点赞日记

personid1 = request.params['personid1']从前端接受点赞人的id， personid2= request.params['personid2']从前端接受被点赞人的id， request.params['diaryid']从前端接受日记id， request.params['liketime']从前端接受点赞时间，将相关数据在点赞表中进行创建。并在个人详情表中，对对应个人id的点赞数进行调整。

```

#点赞日记
def DiaryLike(request):

    request.params = json.loads(request.body)
    personid1 = request.params['personid1']
    personid2= request.params['personid2']
    diaryid = request.params['diaryid']
    liketime=request.params['liketime']

```

```

LikeList.objects.create(personid1=personid1, personid2=personid2, diaryid=diaryid
, liketime=liketime )
global Person2Change

#个人资料里的点赞数(主动)加1
Person2Change = Personel.objects.get(personid=personid1)
Person2Change.likenum = Person2Change.likenum + 1
Person2Change.save()

return JsonResponse({'ret': 0, 'msg': '点赞成功'})

```

4.取消点赞

personid1 = request.params['personid1']从前端接受点赞人的id, personid2= request.params['personid2']从前端接受被点赞人的id, request.params['diaryid']从前端接受日记id, 相关数据在点赞表中进行删除。并在个人详情表中, 对对应个人id的点赞数进行调整。

```

#取消点赞
def CancelDiaryLike(request):
    request.params = json.loads(request.body)
    #点赞人id
    personid1 = request.params['personid1']
    #日记id
    diaryid = request.params['diaryid']

    global LikeListChange
    global Person2Change

    # 个人资料里的点赞数(主动)减1
    Person2Change = Personel.objects.get(personid=personid1)
    Person2Change.likenum = Person2Change.likenum -1
    Person2Change.save()

    LikeListChange = LikeList.objects.get(personid1=personid1, diaryid=diaryid)
    LikeListChange.delete()

    return JsonResponse({'ret': 0, 'msg': '取消点赞成功'})

```

5.评论日记

request.params['personid1']得到评论人的id, personid2= request.params['personid2']得到被评论人的id, diaryid = request.params['diaryid']得到被评论的日记id, commenttime=request.params['commenttime']得到评论产生的时间, commentcontent=request.params['commentcontent']得到评论的内容, 将这五个参数在数据表中进行建立, 同时, 对于评论人, 用Personel.objects.get(personid=personid1)找到对应的个人详情表, 对评论数进行调整

```

#评论日记
def DiaryComment(request):

    request.params = json.loads(request.body)
    personid1 = request.params['personid1']
    personid2= request.params['personid2']

```

```

diaryid = request.params['diaryid']
commenttime=request.params['commenttime']
commentcontent=request.params['commentcontent']

CommentList.objects.create(personid1=personid1, personid2=personid2, diaryid=diaryid, commenttime=commenttime, commentcontent=commentcontent)

global Person2Change

#个人资料里的评论数(主动)加1
Person2Change = Personel.objects.get(personid=personid1)
Person2Change.commentnum = Person2Change.commentnum + 1
Person2Change.save()

return JsonResponse({'ret': 0, 'msg': '评论成功'})

```

6.删除评论

request.params['personid1']得到评论人的id, personid2= request.params['personid2']得到被评论人的id, diaryid = request.params['diaryid']得到被评论的日记id, commenttime=request.params['commenttime']得到评论产生的时间, 依据这五个参数删除对应评论表里的内容, 同时, 对于评论人, 用Personel.objects.get(personid=personid1)找到对应的个人详情表, 对评论数进行调整

```

#删除评论
def DeleteDiaryComment(request):
    request.params = json.loads(request.body)
    #点赞人id
    personid1 = request.params['personid1']
    #日记id
    diaryid = request.params['diaryid']
    #日记评论时间
    commenttime=request.params['commenttime']

    global CommentListChange
    global Person2Change

    # 个人资料里的评论数(主动)减1
    Person2Change = Personel.objects.get(personid=personid1)
    Person2Change.commentnum = Person2Change.commentnum -1
    Person2Change.save()

    CommentListChange =
    CommentList.objects.get(personid1=personid1, diaryid=diaryid, commenttime=commenttime)
    CommentListChange.delete()

return JsonResponse({'ret': 0, 'msg': '删除评论成功'})

```

7.4个人详情页面模块的实现

1. 创建用户详情

创建详情功能是在用户第一次登陆时，对有关资料进行的填写，`request.params['personid']`得到个人id，`request.params['personname']`得到个人姓名，`request.params['personsing']`得到个人签名，用if语句对个人姓名进行判断，若空则向前端提示错误信息，若不为空则在个人详情表里创建相关内容。

```
# 创建用户详情（用于用户第一次登录时，对有关资料的填写）
def NewPersonel(request):
    request.params = json.loads(request.body)
    personid = request.params['personid']
    personname = request.params['personname']
    personsing = request.params['personsing']
    if personname:

        Personel.objects.create(personid=personid, personname=personname, personsing=personsing)
        return JsonResponse({'ret': 0, 'msg': '创建个人详情成功'})
    else:
        return JsonResponse({'ret': 1, 'msg': '用户名不能为空！'})
```

2. 获取用户详情

获取用户详情列表先用`request.GET.get('personid')`得到个人id，之后对数据库进行各个字段的查找，将对应的数据返回至前端

```
# 获取用户详情
def GetPersonel(request):
    personid=request.GET.get('personid')

    personname=Personel.objects.values("personname").filter(personid=personid)
    data1 = list(personname)
    title = data1[0]['personname']
    print(title)

    personsing = Personel.objects.values("personsing").filter(personid=personid)
    data11 = list(personsing)
    title1 = data11[0]['personsing']
    print(title1)

    follownum= Personel.objects.values("follownum").filter(personid=personid)
    data12 = list(follownum)
    title2 = data12[0]['follownum']
    print(title2)

    fansnum = Personel.objects.values("fansnum").filter(personid=personid)
    data13 = list(fansnum)
    title3 = data13[0]['fansnum']
    print(title3)

    commentnum = Personel.objects.values("commentnum").filter(personid=personid)
    data14 = list(commentnum)
    title4 = data14[0]['commentnum']
    print(title4)

    likenum = Personel.objects.values("likenum").filter(personid=personid)
    data142 = list(likenum)
```

```

title42 = data142[0]['likenum']
print(title42)

return JsonResponse({'ret': 0,
                    'personname': title,
                    'personsing': title1,
                    'follownum': title2,
                    'fansnum': title3,
                    'commentnum': title4,
                    'likenum': title42,
                    })

```

3.获取点赞列表

获取点赞列表用request.GET.get('personid2')从前端得到个人id，根据个人id，将评论表中所有包含该id的字段全都过滤出来，打包成data数据发送至前端。

```

#获取点赞列表
def GetPersonelLike(request):
    personid2= request.GET.get('personid2')
    data = list(LikeList.objects.values().filter(personid2__contains=personid2))
    #终端查看数据
    print(data)
    return JsonResponse({'ret': 0,
                        'data':data})

```

4.获取评论列表

获取评论列表用request.GET.get('personid2')从前端得到个人id，根据个人id，将评论表中所有包含该id的字段全都过滤出来，打包成data数据发送至前端。

```

#获取评论
def GetPersonelComment(request):
    personid2= request.GET.get('personid2')
    data =
    list(CommentList.objects.values().filter(personid2__contains=personid2))
    print(data)
    return JsonResponse({'ret': 0,
                        'data':data})

```

5.修改用户详情

修改用户详情通过得到需要修改的三个参数，通过个人id在数据表中找到对应的数据后，进行数据的更改，并向前端提示修改成功的信息。

```

# 修改用户详情
def ChangePersonel(request):
    global PersonChange

    request.params = json.loads(request.body)
    personid = request.params['personid']
    personname = request.params['personname']
    personsing = request.params['personsing']

    PersonChange = Personel.objects.get(personid=personid)

```

```
if personname:  
    PersonChange.personname = personname  
if personsign:  
    PersonChange.personsign = personsign  
PersonChange.save()  
  
return JsonResponse({'ret': 0, 'msg': '修改成功'})
```

八、系统测试[陈宇航]

这部分主要讲系统的测试方案，不要简单贴图，要写出具体的自动测试方案。可以前后端分开来写。

8.1 单元测试

功能	测试用例	预期结果	说明
用户登录	用户名: cyh 密码: 123	登陆成功	
	用户名: cyh 密码: 321321321	登录失败	用户名或密码错误
获取用户详情	用户id: 111	获取用户详情成功	
修改用户信息	用户id: 111 用户名: cyh 用户密码: 123	修改用户详情成功	
	用户id: 11 用户名: cyh 用户密码: 123	修改用户详情失败	用户id不存在
	用户id: 11 用户名: cyh 用户密码: ""	修改用户详情失败	用户名或密码为空
用户注册	用户id: 111 用户名: cyh 用户密码: 123	用户注册成功	
	用户id: 11 用户名: cyh 用户密码: ""	用户注册失败	用户名或密码为空
新建日记	日记id: 1 日记标题: "我的第一篇日记" 日记内容: "大家好"	新建日记成功	
	日记id: 1 日记标题: "" 日记内容: ""	新建日记失败	日记或标题内容不能为空
日记上锁	上锁id: 1 上锁密码: 123	日记上锁成功	
	上锁id: 1 上锁密码: ""	日记上锁失败	上锁密码不能为空
日记解锁	日记id: 1 上锁密码: 123	日记解锁成功	
	日记id: 1 上锁密码: 111	日记解锁失败	上锁密码输入有误
获取日记详情	日记id: 1	获取日记详情成功	
删除日记	日记id: 1 日记是否上锁: 0	删除日记成功	
	日记id: 1 日记是否上锁: 1	删除日记失败	需要先输入日记密码

功能	测试用例	预期结果	说明
日记提醒	日记id: 11 日记提醒时间: "2022-6-9 15: 30"	设置日记提醒时间成功	
取消日记提醒	日记id: 11	取消日记提醒时间成功	
日记按日期搜索	日记id: 11 搜索起始日: "2022-5-20" 搜索结束日: "2022-6-10"	日记按日期搜索成功	
日记按关键字搜索	日记id: 11 搜索内容: "莎士比亚"	日记按关键字搜索成功	
关注用户	关注人id: 11 被关注人id: 111	关注用户成功	
取消关注用户	关注人id: 11 被关注人id: 111	取消关注用户成功	
查看热门日记详情	日记id: 11	查看热门日记详情成功	
日记点赞	日记id: 11 点赞人id: 11 被点赞人id: 22 点赞时间: "2022-6-9 15: 30"	日记点赞成功	
取消日记点赞	日记id: 11 点赞人id: 11 被点赞人id: 22	取消日记点赞成功	
日记评论	日记id: 11 评论人id: 11 被评论人id: 22 评论时间: "2022-6-9 15: 30"	日记评论成功	
删除日记评论	日记id: 11 评论人id: 11 被评论人id: 22	删除日记评论成功	
修改个人资料	个人id: 11 个人姓名: cyh 个性签名: 111	修改个人资料成功	
查看获赞列表	个人id: 11	查看获赞列表成功	
查看获评列表	个人id: 11	查看获评列表成功	
创建用户详情	个人id: 11 个人姓名: cyh 个性签名: 111	创建用户详情成功	

8.2 集成测试

功能	测试流程	预期结果
用户登录	前端在登录界面输入正确的用户名和密码	后端校验通过后，提示前端可以登录的信息，前端提示登录成功信息，并跳转至主界面
	前端在登录界面输入错误的用户名和密码	同过后端校验提示错误信息，前端收到后提示用户名或密码错误
获取用户详情	用户输入正确的用户id，进行密码找回	后端依据用户id查询到对应的用户密码，发送至前端显示
用户注册	用户输入正确的id、用户名、密码	后端创建相关的数据库字段，并提示前端注册成功
新建日记	用户用新增按钮输入日记内容后，用保存键进行保存	后端创建相关的数据库字段，并提示前端日记保存成功
日记上锁	用户在前端选择上锁按钮，并输入密码	后端的上锁表建立相关信息，前端提示日记上锁成功
日记解锁	用户在前端输入密码	后端校验密码正确性，前端提示日记解锁成功
获取日记详情	用户点击某一个日记	后端返回日记详情，前端提示获取日记详情成功
删除日记	用户长按日记框图选择删除	后端删除对应数据库信息，前端提示删除日记成功
日记提醒	用户在界面设置日记提醒时间	后端增加数据库信息，前端提示设置日记提醒时间成功
取消日记提醒	日在界面上选择取消日记提醒时间	后端删除对应数据库信息，前端提示取消日记提醒时间成功
日记按日期搜索	用户在日期框选择日期，并进行搜索	后端通过日期范围进行数据库查找，前端提示日记按日期搜索成功
日记按关键字搜索	用户在搜索框输入内容，并进行搜四	后端通过关键字进行数据库查找，前端提示日记按关键字搜索成功
关注用户	用户点击关注按钮	后端增加数据库信息，后前端提示关注用户成功
取消关注用户	用户点击关注按钮	后端删除对应数据库信息，前端提示取消关注用户成功
查看热门日记详情	用户点击热门日记框图	后端增加数据库信息，前端提示查看热门日记详情成功
日记点赞	用户在点赞日记	后端增加数据库信息，前端提示日记点赞成功
取消日记点赞	用户取消日记点赞	后端删除对应数据库信息，前端提示取消日记点赞成功
日记评论	用户在评论框输入内容，并点击发送	后端增加数据库信息，前端提示日记评论成功
修改个人资料	用户在编辑個人資料进行修改	后端将数据库信息进行修改，前端提示修改个人资料成功

功能	测试流程	预期结果
查看获赞 列表	用户点击获赞数	后端依据id返回数据，前端提示查看获赞列表成功
查看获评 列表	用户点击获评数	后端依据id返回数据，前端提示查看获评列表成功
创建用户 详情	用户在初次登陆时填写相关资 料	相关资料信息传送至后端，数据库创建后提示创建 成功的信息

8.3 测试部署及结果

整个系统的后端测试，我主要部署在了GitHub上，利用自带的git action，在每一次提交后端代码时，都会自动的进行测试，来确保代码的正确率。

(1) git action部署文件

如下图是对git action的环境进行部署，主要的关键点在于要将requirements文件放在和workflow同一个目录下，同时执行test测试的路径要对，例如我这里的 `python django/memodjango/manage.py test` 命令

```

1 name: Django CI
2
3 on:
4   push:
5     branches: [ "main" ]
6   pull_request:
7     branches: [ "main" ]
8
9 jobs:
10   build:
11
12     runs-on: ubuntu-latest
13     strategy:
14       max-parallel: 4
15       matrix:
16         python-version: [3.7, 3.8, 3.9]
17
18     steps:
19       - uses: actions/checkout@v3
20       - name: Set up Python ${{ matrix.python-version }}
21         uses: actions/setup-python@v3
22         with:
23           python-version: ${{ matrix.python-version }}
24       - name: Install Dependencies
25         run: |
26           python -m pip install --upgrade pip
27           pip install -r requirements.txt
28       - name: Run Tests
29         run: |
30           python django/memodjango/manage.py test

```

(2) 测试核心

对于测试，我认为主要有两类：一类是必定成功的，例如查看点赞列表、评论列表之类的接口，对于这类接口，主要是对返回的数据加以验证，如下图（以获取点赞列表为例）：

对于这部分接口逻辑，我将得到的数据在终端进行打印，通过和数据库中的数据进行对比，来校验数据是否有误，数据格式是否符合标准

```

#获取点赞列表
def GetPersonelLike(request):
    personid2= request.GET.get('personid2')
    data = LikeList.objects.values().filter(personid2__contains=personid2)
    #终端查看数据
    print(data)
    return JsonResponse({'ret': 0,
                        'data':data})

```

```
# 获取点赞列表单元测试
def test_GetPersonelLike(self):
    # 成功实例
    r = requests.get(self.GetPersonelLike_url + '?personid2=' + "11")py
    result1 = r.json()
    self.assertEqual(result1['ret'], 0)
```

而另一类接口，例如登录、注册之类的接口，它们是可能由于用户名密码错误等问题不成功的，那么为了让前端知道为什么，这类接口需要写正反两个例子（成功和失败），如下图（以用户登录为例）：

```
# 用户登录
def userLogin(request):
    username = request.GET.get('username')
    passwd = request.GET.get('passwd')
    type = User.objects.values().filter(username=username, passwd=passwd)
    if type:
        return JsonResponse({'ret': 0, 'msg': '登录成功'})
    else:
        return JsonResponse({'ret': 1, 'msg': '登陆失败'})
```

```
用户登录单元测试
def test_UserLogin(self):
    # 成功实例
    r = requests.get(self.UserLogin_url + '?username=' + "cyh" + '&passwd=' +
    "123")
    result1 = r.json()
    self.assertEqual(result1['ret'], 0)

    # 失败实例
    q = requests.get(self.UserLogin_url + '?username=' + "cyh" + '&passwd=' +
    "123456")
    result2 = q.json()
    self.assertEqual(result2['ret'], 1)
```

(3) 测试结果

由于我一般会在pycharm上先用python manage.py test命令进行测试，因此提交到GitHub上通过git action进行

测试的正确率基本为100%，见下图（部分展示，具体可见GitHub上的aciton页面）：

24 workflow runs			
	Event ▾	Status ▾	Branch ▾
Actor ▾			
✓ 删除日记评论功能逻辑实现和测试 (后端) Django CI #27: Commit 36008c3 pushed by jumpshotsniper	main	4 hours ago 1m 16s	...
✓ 日记评论功能逻辑实现和测试 (后端) Django CI #26: Commit 7be6887 pushed by jumpshotsniper	main	4 hours ago 32s	...
✓ 取消日记点赞功能逻辑实现和测试(后端) Django CI #25: Commit c839e35 pushed by jumpshotsniper	main	9 hours ago 29s	...
✓ 日记点赞功能逻辑实现和测试 (后端) Django CI #24: Commit e2a3d9f pushed by jumpshotsniper	main	9 hours ago 29s	...
✓ 取消关注功能逻辑实现和测试 (后端) Django CI #23: Commit 34ab229 pushed by jumpshotsniper	main	22 hours ago 33s	...
✓ 关注用户重新进行git action自动测试 (后端) Django CI #22: Commit eb694f3 pushed by jumpshotsniper	main	23 hours ago 27s	...
✓ 修改個人資料功能逻辑实现和测试 (后端) Django CI #21: Commit 606eb43 pushed by jumpshotsniper	main	23 hours ago 29s	...
✓ 查看评论列表功能逻辑实现和测试(后端) Django CI #20: Commit 4cb1fe7 pushed by jumpshotsniper	main	yesterday 33s	...

九、系统部署 [陈宇航]

本次系统部署方面，主要依据网上的教程，将后端部署在了docker上

1.docker介绍

docker概念

docker是一种容器技术，用来创造独立的软件容器，属于操作系统的虚拟化技术，隔离的进程独立于宿主和其他隔离的进程。

docker容器对比虚拟机

虚拟机：需要有必要的二进制库和完整的操作系统，并且占据所有预分配的资源。

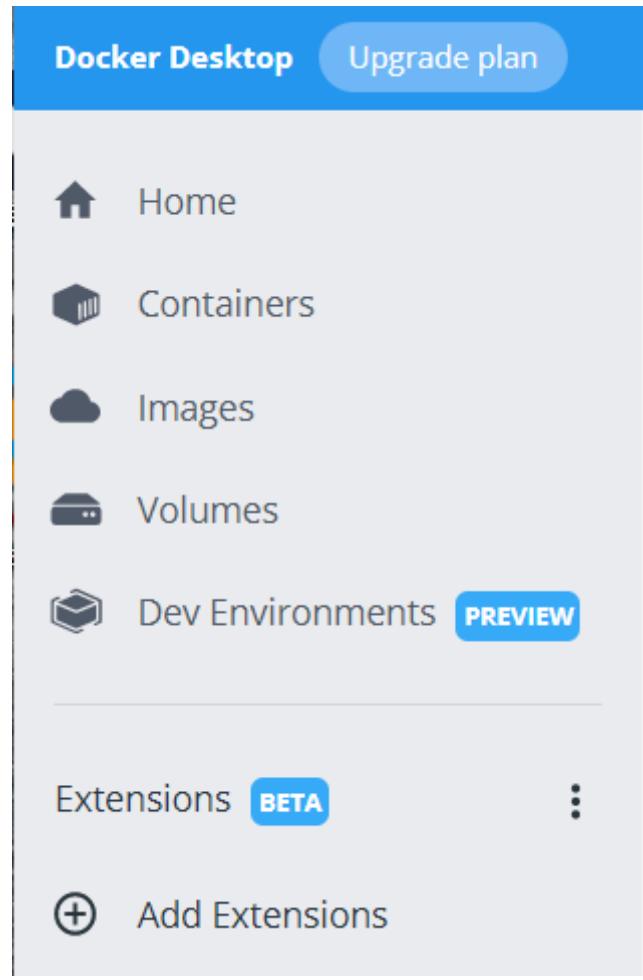
docker容器：与宿主共享操作系统，可以实现资源的动态分配，在用户空间以分离的进程运行

docker镜像、容器和仓库

镜像是文件，容器是进程；他们之间的关系像是类和实例化对象的关系，一个类可以有多个实例化对象。而仓库是一个其中的存储、分发镜像的服务。docker pull是从仓库下载docker镜像到本地，docker run是创建一个容器并运行一个命令，docker exec -it是在运行的容器中执行命令

2.部署流程

1.下载docker



2. 使用命令 `docker run --name my-nginx -p 80:80 -d nginx:latest`，创建一个容器，命令为my-nginx，`-d` 表示后台运行并返回容器的ID，`-p` 将端口指定到宿主的80端口

3. 在django项目根目录下创建 `Dockerfile` 文件，该文件用于生成docker镜像，内容如下

```
Project: memodjango [bymys]
  + byms
  + common
  + diary
  + personel
  + public_diary
  + User
    + db.sqlite3
    + Dockerfile
    + manage.py
    + pip
    + requirements.txt
  + External Libraries
  + Scratches and Consoles

Dockerfile
1 # 建立 python3.7 环境
2 FROM python:3.8
3
4 # 镜像作者
5 MAINTAINER LALALA
6
7 # 设置 python 环境变量
8 ENV PYTHONUNBUFFERED 1
9
10 # 设置pip源为国内源
11 COPY pip.conf /root/.pip/pip.conf
12
13 # 在容器内创建mysite文件夹
14 RUN mkdir -p /var/www/html/mysite
15
16 # 设置容器内工作目录
17 WORKDIR /var/www/html/mysite
18
19 # 将当前目录文件加入到容器工作目录中 (. 表示当前宿主机目
20 ADD . /var/www/html/mysite
21
22 # pip安装依赖
23 RUN pip install -r requirements.txt
```

The screenshot shows a code editor with a sidebar containing a project structure for a Django application named "memodjango". The "Dockerfile" tab is selected, showing the following Dockerfile content:

4. 继续在项目根目录下创建 `pip.conf` 文件，用于修改pip镜像源

```
[global]
index-url = https://pypi.tuna.tsinghua.edu.cn/simple
[install]
trusted-host=mirrors.aliyun.com
```

5. 创建 requirements.txt 文件，内容为 django 的版本

6. 在项目根目录输入命令 docker build -t django_docker_img:v1 . 创建镜像，使用命令 docker images 查看镜像

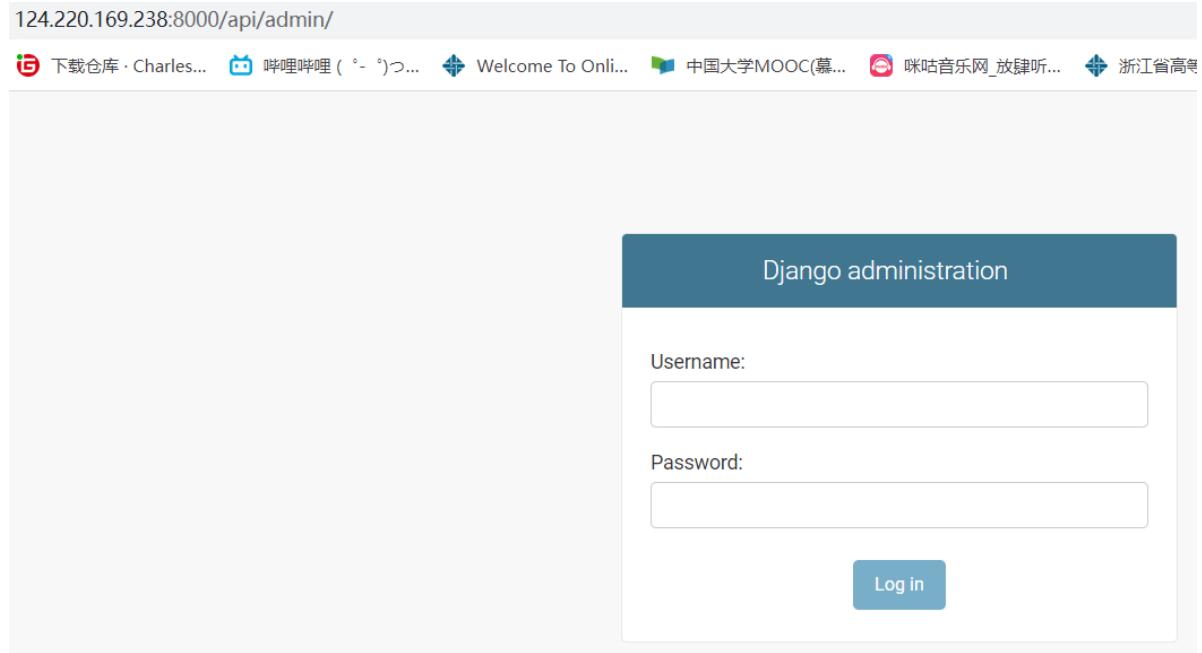
7. 输入命令 docker run -it -d -p 80:8000 --name mysite django_docker_img:v1 创建并运行容器，此命名将该容器映射到本地 80 端口，使用 docker ps 查看运行情况

8. 执行数据库迁移命令

```
python manage.py makemigrations
python manage.py migrate
python manage.py runserver 0.0.0.0:8000
```

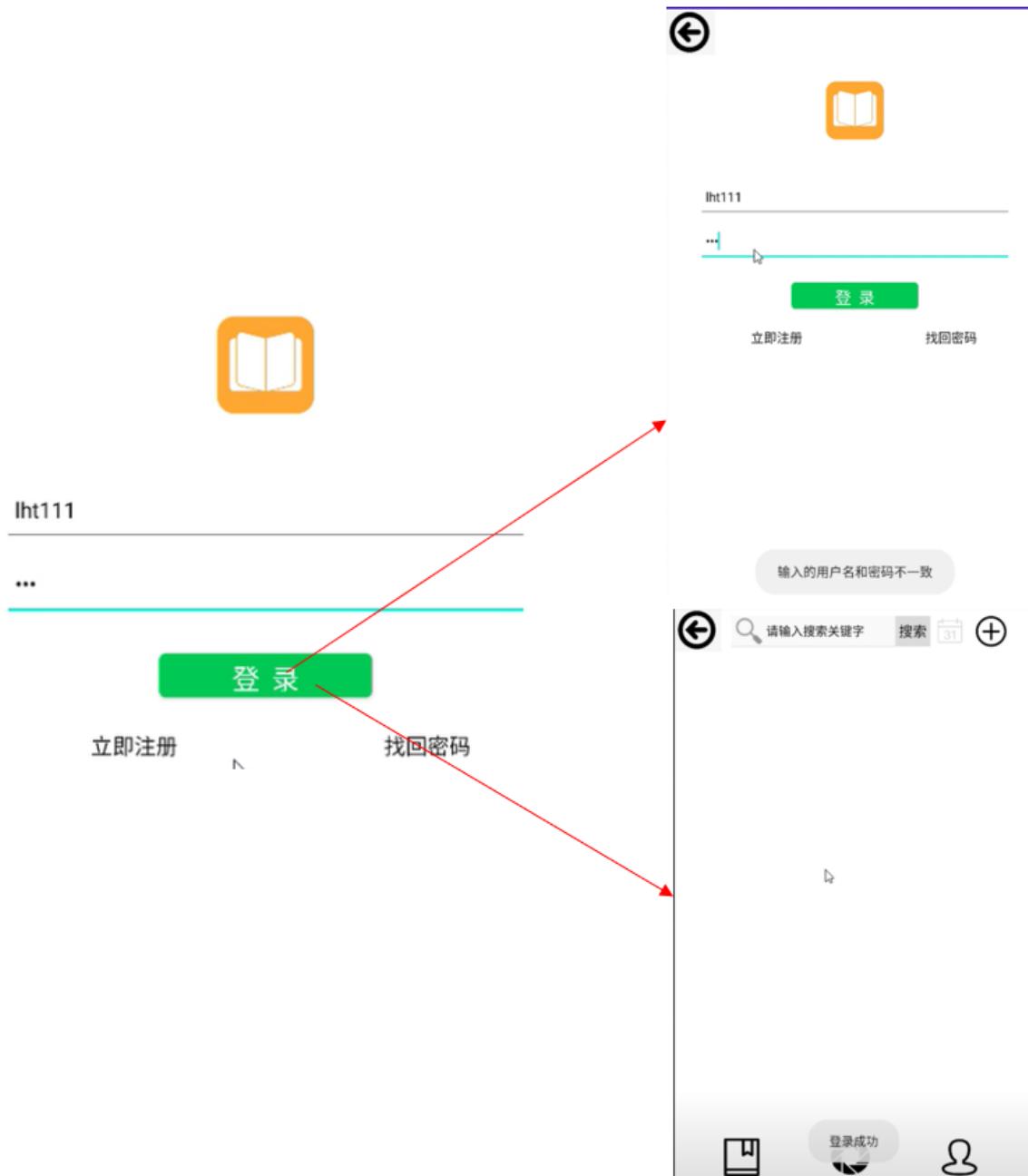
3. 部署结果

本地访问成功截图展示

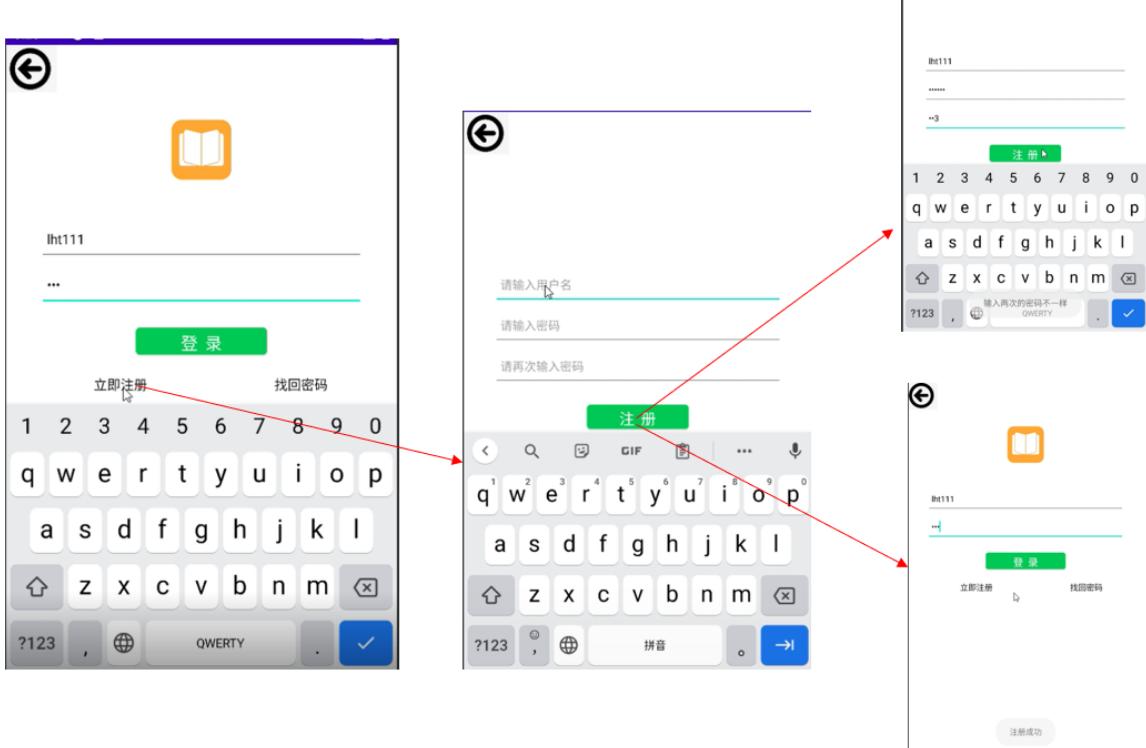


十、功能展示 [陈宇航, 李浩腾]

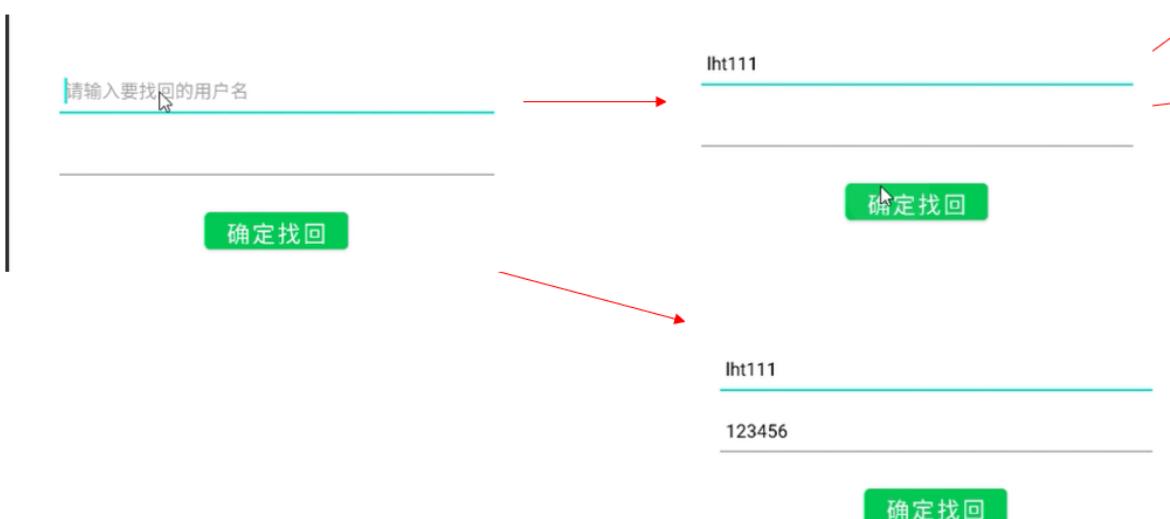
1. 登录功能



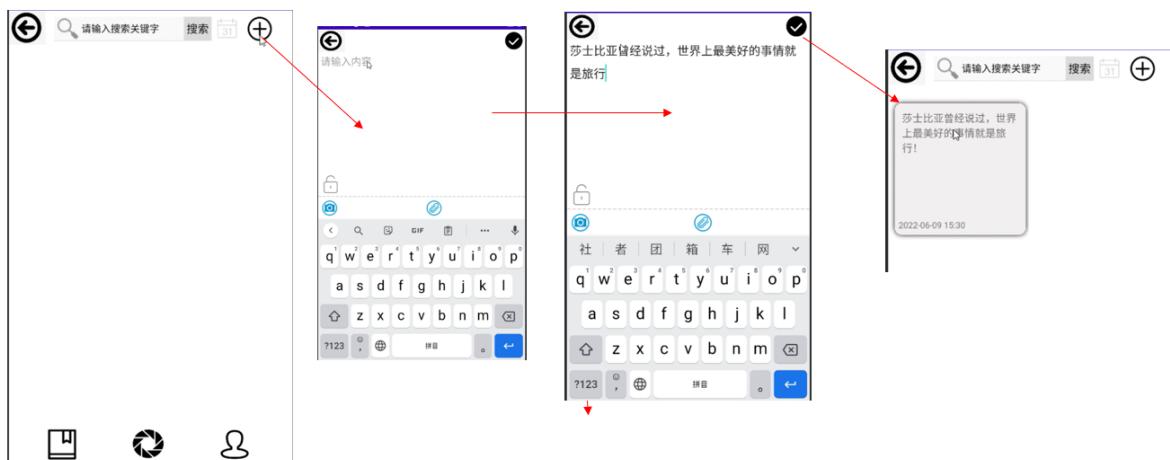
2. 注册功能



3. 找回密码功能



4. 新增日记功能



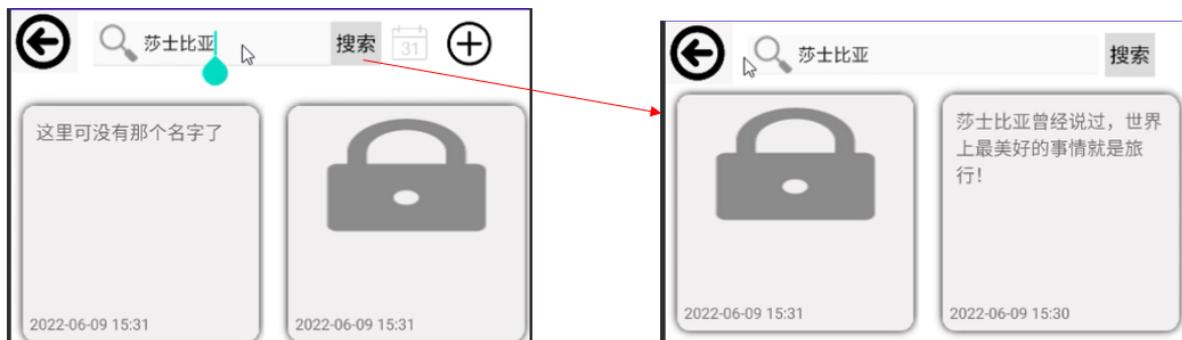
5. 日记上锁



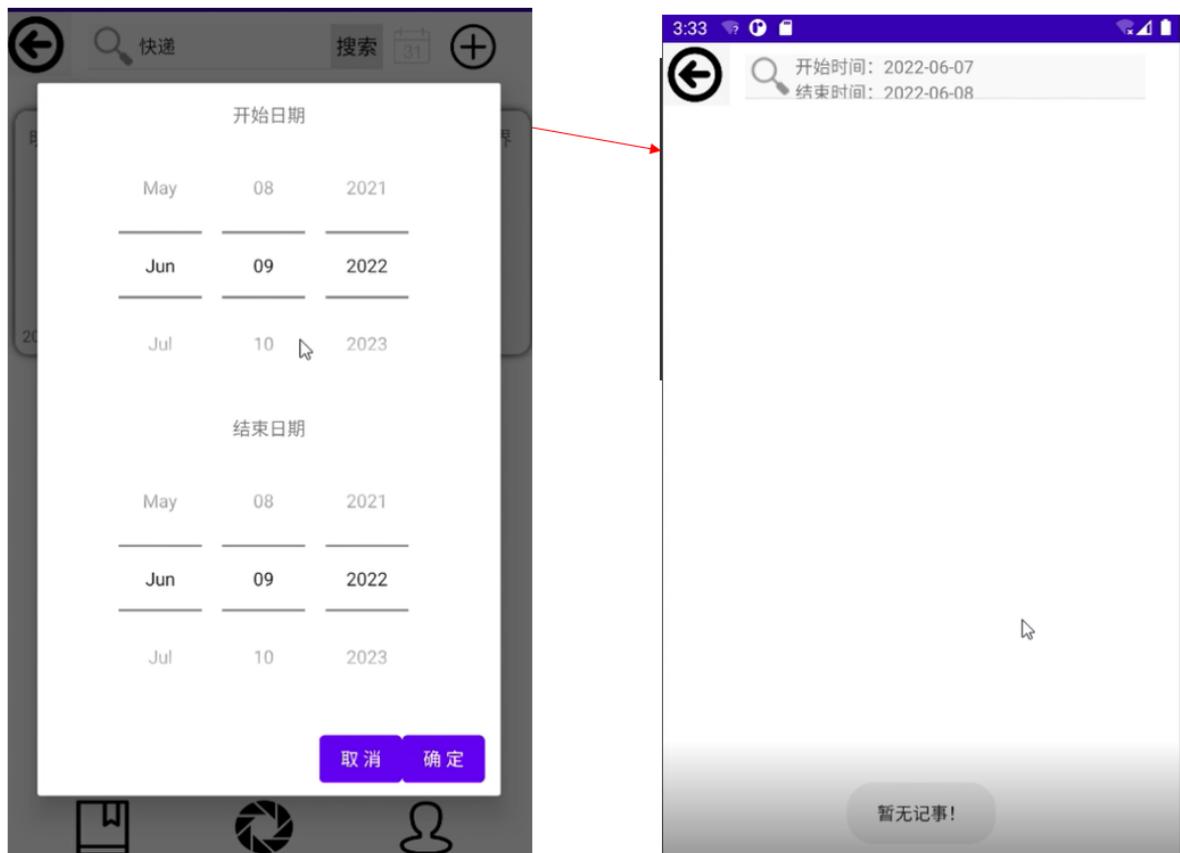
6. 日记解锁



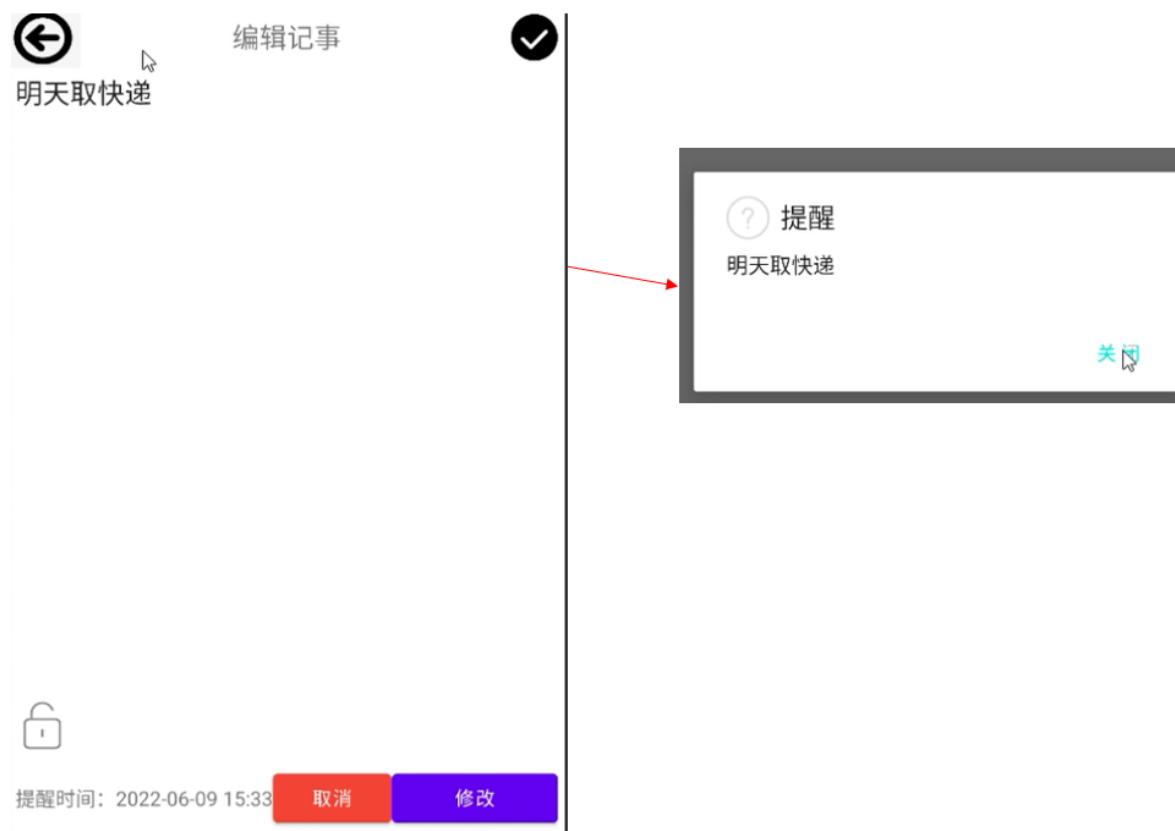
7.按内容搜索



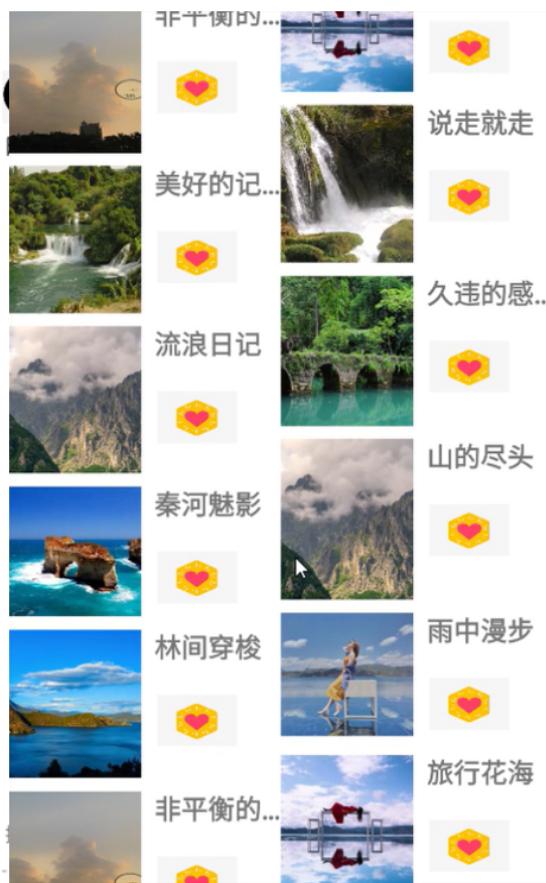
8.按日期搜索



9.日记提醒



10.热门日记圈子



日记分享

李浩腾 发布于昨天

旅行花海

说走就走的旅行，要么缘由幸福稳定和宽裕，要么祸起无力无奈和逃避。在旅途中，我遇见了你，你我相识是缘分！看着你手中的戒指，你说，你可以把它取下来吗？当我要取的时候，你淘气的躲开了，你说只有有缘人才可以取下，我看着你手中的戒指，想做你的有缘人，可是我知道结果是惨淡的，但还是心存希望



留言问问更多细节吧~

评论 点赞

日记分享

发布于昨天

旅行花海

说走就走的旅行，要么缘由幸福稳定和宽裕，要么祸起无力无奈和逃避。在旅途中，我遇见了你，你我相识是缘分！看着你手中的戒指，你说，你可以把它取下来吗？当我要取的时候，你淘气的躲开了，你说只有有缘人才可以取下，我看着你手中的戒指，想做你的有缘人，可是我知道结果是惨淡的，但还是心存希望

作者写得很好

评论 点赞

的 | 看 | 听 | 吗 | 办 | 打 |
q w e r t y u i o p
a s d f g h j k l
z x c v b n m
?123 , . 拼音 ←



发布于浙江省长青山 浏览 124人

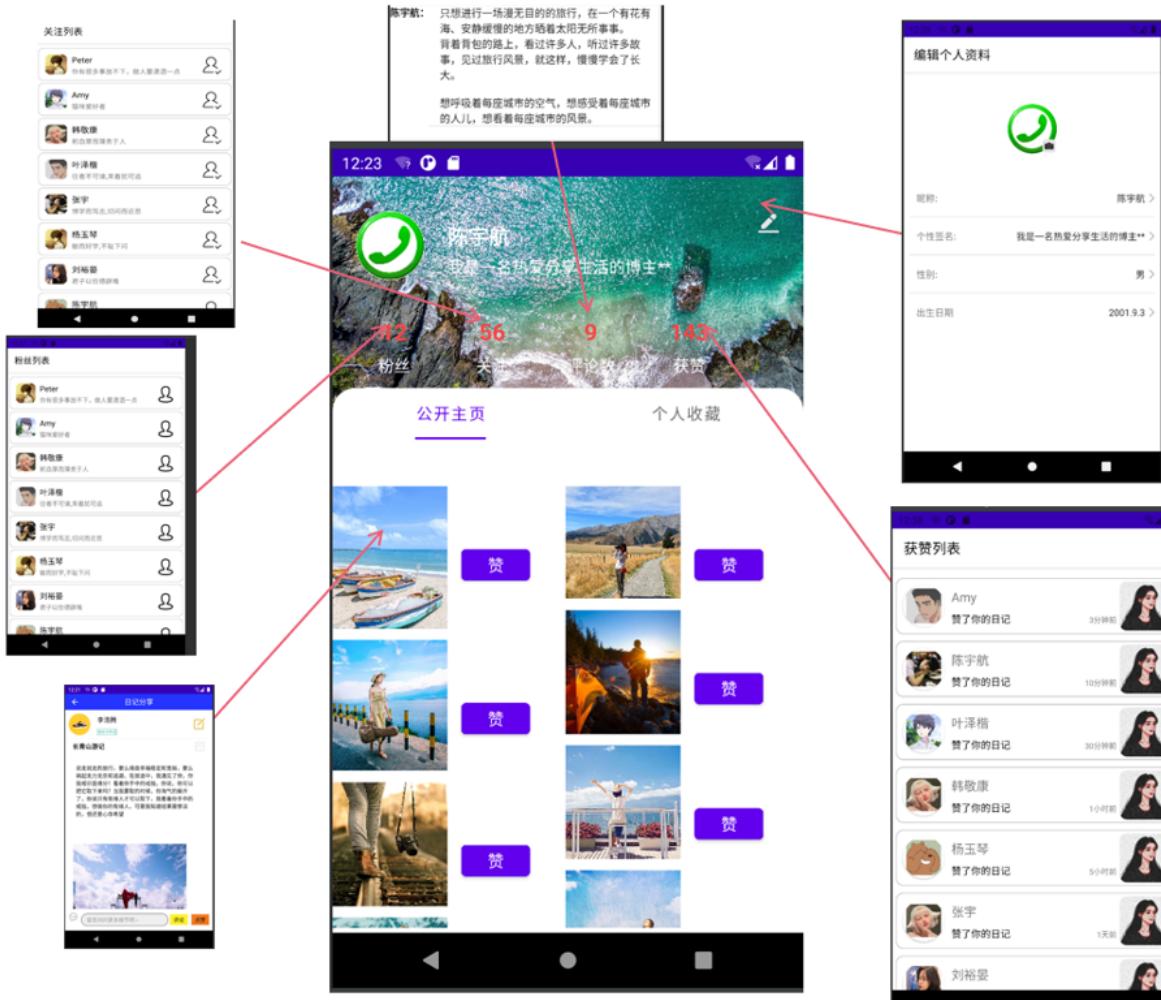
全部评论 • 1

陈宇航： 作者写得很好

留言问问更多细节吧~

评论 点赞

11.个人用户详情模块



十一、清单[陈宇航，李浩腾]

这部分列出项目提交的清单，如：

- 前端代码: app目录
- 后端代码: django目录
- 项目过程文件: design目录
- 项目最终文档(pdf和md): finaldoc/doc
- 项目最终评审ppt: finadoc/ppt
- 项目演示视频: finaldoc/video

十二、总结 [陈宇航，李浩腾]

陈宇航

这次的项目，我基本参与了整个流程开发，绝大部分任务都有我的参加，不同于以往的实验课，这次实验课的开发，让我切身体会到了项目开发的规范。以前我总是习惯于做一点功能加一点功能，而通过这次实验课，我知道了项目开发前，应先定好整个系统的框架与实现技术，之后是对功能需求进行分析，列出功能导图后，可以先进行原型的设计。然后是去设计API接口，这里的接口一定要规范，基于AP接口，后端和前端可以分别同步的进行开发，并利用APIfox进行测试，这样最后进行前后端联调时，每一部分的连接由于测试过了，都是完全规范的。同时我也掌握了很多工具的使用，利依托github管理，部署git action进行自动测试

对于这次的项目，我认为后端实现的较好，整体功能和测试都很完备，且遵循了接口设计的规范。前端方面由于很多技术原因，有些功能和后端的整体联通还有很大问题，有些功能目前只局限于能在前端实现。还有一个遗憾就是由于时间原因，没能完成系统部署。

李浩腾

在这学期的移动软件开发中，我发现了自身存在的许多问题，也收获到了很多，学到了很多开发方面的知识。在项目开发中项目一开始就应该让后台和数据不分离，我认为如果两者分开的话，在维护测试的时候有很大的问题。这一条有待研究，但是我觉得这么分离不是很好，但是也有可能是我的经验比较少，需要我日后慢慢理解。

同时项目缺少阶段性测试，导致所有模块做完之后，导致需要修很久以前的bug，耗费时间长。需求从一开始就一直在变化，我们一开始没有一个明确的项目规划，在已经完成一部分的情况下临时又多加了许多功能，导致项目变得很混乱。

虽然遇到了很多问题，但是正是这些问题在推动我成长，比如文档的规范化，和小组的合作开发等等都是在这学期中新学到的，如果是一个模板文档，那么这个模板文档应该经得起考验，应该让大家根据这个模板产生的文档的格式是统一的，不能在模板文档里面都存在不明确。还有很多地方等着我继续学习继续改正。

十二、参考文献 [陈宇航，李浩腾]

系统所参考的文献或者代码，比如：

- 安卓瀑布流布局: https://github.com/dodola/android_waterfall
- 白月黑羽django学习:[白月黑羽 \(byhy.net\)](#)
- 安卓控件库: <https://github.com/Ccapton/Collection-Android-Progress>