微信抢票应用交付文档

项目Github地址

hongfz16/WeChatTicket

小组成员

- 王泽宇
  - 学号：2016013258
  - 邮箱：ycdfwzy@outlook.com
- 李仁杰
  - 学号：2016013271
  - 邮箱：ShadowIterator@hotmail.com
- 张佳麟
  - 学号：2016013256
  - 邮箱：sherlockcooper@sina.com
- 洪方舟
  - 学号：2016013259
  - 邮箱：hongfz16@163.com

交付产品

测试公众号二维码

后台管理员

- 账号：admin
- 密码：thssojadmin

产品目标

产品功能目标

本产品的核心功能主要包含两个部分，一个部分为管理员后台管理系统，主要负责管理员管理活动的操作，包括但不限于活动的创建，活动的修改，活动的删除，活动检票，调整微信公众号菜单；另一个部分为微信公众号，主要负责用户对活动门票的操作，包括但不限于用户的抢票，查看活动信息，查看门票，退票。

产品性能目标

本产品对性能方面有一定的要求，主要的要求在于微信公众号相关的API，用户抢票API需要能够承受一定程度的并发量；微信公众号相关的操作由于需要与微信服务器交互，因此有严格的时限，对实现方式的性能有一定的要求。

设计实现

本次持续集成实验需要实现的接口主要有两部分，一部分为继承自 `APIView` 类的接口，另一部分为继承自 `WeChatHandler` 以处理微信消息的接口。

继承APIView类的接口

为了优化Ticket表的查询操作，将原先的外键 `activity` 改为整数 `activity_id` 。

- 1 /api/u/user/bind

  与原有接口实现相同，加入学号为10位的判断

- 2 /api/u/activity/detail
  - GET
  - 判断能否通过传入参数获得对应活动
  - 判断活动状态是否为已发布
  - 条件均满足则按照要求返回活动详细信息

- 3 /api/u/ticket/detail
  - GET
  - 通过给定openid查找用户是否存在
  - 通过给定uniqueid查找电子票是否存在
  - 对比用户和票据中student_id是否一致
  - 条件均满足则返回电子票详细信息

- 4 /api/a/login
  - GET
  - 检查登录状态
  - 登录则正常返回，否则返回非0错误代码
  - POST
  - 检查输入用户名密码是否匹配
  - 匹配则使该用户登录，否则返回非0错误代码

- 5 /api/a/logout
  - POST
  - 检查当前是否处于用户登录状态
  - 是则登出该用户，否则返回非0错误代码

- 6 /api/a/activity/list
  - GET
  - 判断是否登录，未登录则抛出异常。
  - 访问数据库，查找所有未删除的活动 `STATUS_STATUS>0`

- 将查找到的活动按照要求格式返回。

- 7 /api/a/activity/delete
  - POST
  - 判断是否登录，未登录则抛出异常。
  - 在数据库中尝试删除特定 `id` 的活动，若失败则抛出异常。

- 8 /api/a/activity/create
  - POST
  - 检查用户是否登录
  - 检测参数是否完整，参数类型是否正确
  - 根据所给参数信息在数据库中添加新的活动

- 9 /api/a/image/upload
  - POST
  - 将传入图片数据写入static路径下
  - 返回存储图片的URL

- 10 /api/a/activity/detail
  - GET

1. 判断是否登录，未登录则抛出异常。
2. 在数据库中按照 `id` 查找特定活动。
3. 将查找到的活动按照要求格式返回。

难点：已经被预定的票数 `bookedTickets` =总票数 `totalTickets` -剩余票数 `remainTickets`

已经使用的票数 `usedTickets` 可以以活动id为关键字，查询Ticket表中的票 `status=STATUS_USED` 的数量。

  - POST

1. 判断是否登录，未登录则抛出异常。
2. 在数据库中按照 `id` 查找特定活动。
3. 按照文档要求判断修改是否合法，不合法则抛出异常。
4. 修改活动，并保存到数据库。

- 11 /api/a/activity/menu
  - GET
  - 判断是否登录，未登录则抛出异常。
  - 获取所有 `status=STATUS_PUBLISHED` 的活动。

    - 按照文档格式要求返回数据，`menu`项可以通过查询`CustomWeChatView.menu`得知。
    - POST
    - 判断是否登录，未登录则抛出异常。
    - 通过提交的id数组，从数据库中获得所有活动。
    - 利用`CustomWeChatView.update_menu`方法更新数组。

- 12 /api/a/activity/checkin

    - POST
    - 检查输入参数是否完整
    - 检查能否通过actid获得对应活动
    - 检查能否通过ticket或studentId获得对应电子票
    - 检查活动与电子票是否对应
    - 检查电子票是否处于可用状态
    - 检查均通过则返回电子票的id与学生id，并将电子票状态从可用改为已使用

继承WeChatHandler类的接口

微信测试公众号中的消息响应主要通过定义不同的继承自WeChatHandler类的处理来实现。在定义类的check方法和handle方法后，将其加入`CustomWeChatView`的handler列表中即可。除框架已给出handler外，我们自己实现的handler如下：

- 1 BounceHandler

    - check

    检查用户输入信息是否为“退票 xxx”的形式，并以“退票”为关键字进行检查，若符合则返回True。

    - handle
    - 检查用户是否绑定学号
    - 提取用户输入的活动代称部分
    - 检查能否通过活动代称对应活动
    - 对后续过程加锁
    - 检查能否通过活动以及当前用户学号获得对应票据
    - 将票据从数据库中删除
    - 将活动的剩余票数+1
    - 返回成功信息
    - 以上任意操作未通过或失败均返回失败提示

- 2 BookTicketsHandler

    - check

    如果是`text_event`且输入了抢票，那么就从数据库中查找是否有活动的`key`与输入一致；如果是`click_evnet`，那么就遍历`CustomWeChatView.menu`中的订票按钮，查找确定活动类型。如

果check返回值为 `True` ，那么会在类成员变量中加入 `id` 值为活动id。

- handle

1. 判断是否登录，未登录则抛出异常。
2. 根据 `self.id` 查找固定 `id` ，如果未找到则返回错误信息：未找到该活动！
3. 检测提交时间是否在订票开始于结束区间内，如果不是的话则返回错误信息。
4. 检测当前用户是否在该活动中已经抢到了票，如果已经抢到了，则返回错误信息，防止重复抢票。
5. 如果本活动的余票为0，则返回错误信息：没有余票！
6. 余票减少一张，并在Ticket数据库中创建一张票，并返回提醒信息：抢票成功。

难点：为了保证服务器在抢票高峰期（高并发）能够正确操作，于是采用了 `with transaction.atomic()` 定以数据库的原子事务，并用数据库的 `select_for_update` 方法来获取修改数据。

- 3 CheckTicketHandler
  - check

检查用户是否点击了查票按钮或者输入信息“查票”，若符合则返回True。

  - handle
  - 检查当前用户是否已经绑定学号
  - 通过学号查找该用户所有可用电子票
  - 将提取电子票信息加入待返回列表
  - 若列表为空则返回提示，否则通过reply_news返回

- 4 BookWhatHandler
  - check

检查用户是否点击查票按钮或输入信息“抢啥”，若符合则返回True。

  - handle

1. 获取当前时间
2. 在数据库中查找所有状态为已发布，且还未结束的活动
3. 将提取活动信息加入待返回列表
4. 若列表不为空则通过reply_news返回，否则返回对应提示

- 5 TakeTicketHandler
  - check

检查用户输入信息是否为“取票 xxx”的形式，并以“取票”为关键字进行检查，若符合则返回True。

  - handle

1. 检查用户是否绑定学号
2. 提取用户输入的活动代称部分
3. 检查能否通过活动代称对应活动
4. 获取用户拥有的有效票列表
5. 检查电子票列表中是否有对应查询活动的
6. 若存在活动电子票则通过reply_news返回，否则提示未找到票

测试

测试方案

采用 `django.test` 的测试工具 `TestCase` 进行测试。使用这种方法建立的测试可以使用一个简单的 `python3 manage.py test` 命令批量执行， `django` 会自动产生一个临时数据库进行测试并返回测试结果。

测试对象

- 所有前后端接口
- 所有在 `wechat/handler.py` 中定义的各种 `handler`

测试思路

对于各前后端 `api` 接口进行测试，通过用 `django.test.Client()` 伪造前端给后端的 `get` 和 `post` 请求，然后检查后端是否对数据库进行了正确读写，检查后端是否对前端进行了正确的返回。

对于微信转发给后端的请求(即通过 `wechat/handler.py` 中定义的各种 `handler` 处理的请求)，通过构造相应的 `xml` 文件，然后通过 `django.test.Client()` 发送给后端。

测试用例的设计思路

对于被测试对象，其主要测试用例有以下几种 * 被测试对象完成相应操作、正常返回结果 * 被测试对象返回一个错误： - 提交的参数缺失 - 提交的参数类型错误 - 提交的参数过长 - 提交的参数不满足特定约束(比如开始时间必须小于结束时间) - 提交请求时的上下文不满足特定要求(比如用户未绑定、未登录等)

测试用例的举例

学号绑定

url

`/api/u/user/bind`

主要测试点

- `get` 方法
    1. 测试点
        - 请求
            - `openid` 在数据库中 *已存在 已绑定*
        - 期望返回
            - `code` **为** `0`
            - `data` **为** 对应 `student_id`
    2. 测试点
        - 请求
            - `openid` 为空串
        - 期望返回
            - `code` **非** `0`
            - `data` **为** 空串
    3. 测试点
        - 请求:
            - `openid` 在数据库中 *不存在*
        - 期望返回
            - `code` **非** `0`
            - `data` **为** 空串
    4. 测试点
        - 请求:
            - `openid` 在数据库中 *已存在 未绑定*
        - 期望返回
            - `code` **非** `0`
            - `data` **为** 空串
    5. 测试点
        - 请求
            - `openid` *缺失*
        - 期望返回
            - `code` **非** `0`
            - `data` **为** 空串
- `post` 方法

1. 测试点
    - 请求
        - `openid` *已存在 未绑定*
        - `student_id` *合法*
    - 期望返回
        - `code` **为** `0`
    - 期望行为
        - 数据库中对应项的 `student_id` 域 **为** 请求中 `student_id` 域中的内容
2. 测试点
    - 请求
        - `openid` *已存在 未绑定*
        - `student_id` *不合法*
    - 期望返回
        - `code` **非** `0`
    - 期望行为
        - 不对数据库进行任何更改
3. 测试点
    - 请求
        - `openid` *不存在*
        - `student_id` *合法*
    - 期望返回
        - `code` **非** `0`
    - 期望行为
        - 不对数据库进行任何更改
4. 测试点
    - 请求
        - `openid` *已存在 未绑定*
        - `student_id` *缺失*
    - 期望返回
        - `code` **非** `0`
    - 期望行为
        - 不对数据库进行任何更改
5. 测试点

    - 请求
        - `openid` *已存在 已绑定*
        - `student_id` *合法*
    - 期望返回
        - `code` **非** `0`
    - 期望行为
        - 不对数据库进行任何更改

备注

由于组内有一名同学数据库版本问题，创建 `User.objects.create()` 在数据库中新增一行时， `student_id` 域不能为空，于是最终的测试代码中，用随机的字符串代替了空串。

创建活动

url

`/api/a/activity/create`

测试点

- `post` 方法

    1. 测试点
        - 上下文
            - 用户 *未登陆*
        - 请求
            - 所有参数均合法
        - 期望行为
            - 不对数据库做任何更改
        - 期望返回
            - `code` **非** `0`
    2. 测试点
        - 上下文
            - 用户 *已登陆*
        - 请求
            - 所有参数均合法
        - 期望行为

        - 数据库中新增一项，对应域与请求的对应参数一致

    - 期望返回
        - `code` **为** `0`
        - `data` **为** 数据库中对应项的 `id`

3. 测试点
    - 上下文
        - 用户 *已登陆*

    - 请求
        - `startTime` 域大于 `endTime` 域
        - 其余参数合法

    - 期望行为
        - 不对数据库做任何更改

    - 期望返回
        - `code` **非** `0`

4. 测试点
    - 上下文
        - 用户 *已登陆*

    - 请求
        - `bookStart` 域大于 `bookEnd` 域
        - 其余参数合法

    - 期望行为
        - 不对数据库做任何更改

    - 期望返回
        - `code` **非** `0`

5. 测试点
    - 上下文
        - 用户 *已登陆*

    - 请求
        - `bookEnd` 域大于 `startTime` 域
        - 其余参数合法

    - 期望行为
        - 不对数据库做任何更改

    - 期望返回

        - `code` **非** `0`

6. 测试点
    - 上下文
        - 用户 *已登陆*
    - 请求
        - `totalTickets` 小于 `0`
        - 其余参数合法
    - 期望行为
        - 不对数据库做任何更改
    - 期望返回
        - `code` **非** `0`
7. 测试点
    - 上下文
        - 用户 *已登陆*
    - 请求
        - `key` 的长度大于数据库对应域的 `max_length`
        - 其余参数合法
    - 期望行为
        - 不对数据库做任何更改
    - 期望返回
        - `code` **非** `0`
8. 测试点
    - 上下文
        - 用户 *已登陆*
    - 请求
        - `endTime` 没有精确到秒
        - 其余参数合法
    - 期望行为
        - 不对数据库做任何更改
    - 期望返回
        - `code` **非** `0`
9. 测试点
    - 上下文

        - 用户 *已登陆*

    - 请求
        - `status` 为 `2` (正确的取值范围为 `0` 和 `1`)
        - 其余参数合法

    - 期望行为
        - 不对数据库做任何更改

    - 期望返回
        - `code` **非** `0`

10. 测试点
    - 上下文
        - 用户 *已登陆*

    - 请求
        - `name` 和 `description` 为中文

    - 期望行为
        - 数据库中新增一项，对应域与请求的对应参数一致

    - 期望返回
        - `code` **为** `0`
        - `data` **为** 数据库中对应项的 `id`

缺陷汇总

测试脚本的主要问题

- 当进行过一次 `post` 之后，再进行一次无参数的 `post` 时，会导致参数解析的错误
    - 解决方法: `post` 时，可以带上一个无意义参数
- `django.test.Client()` 无法把 `list` 对象作为参数进行请求

测试脚本测试出的主要问题

- 抢票(`BookTicketsHandler`)
    - 未判断抢票开始和结束时间导致可以在抢票开始前或结束后抢到票
    - `check` 函数错误，导致抢未开始的活动的票时，无法返回正确的回答信息('抢票未开始')
- 创建活动(`/api/a/activity/create`)
    - 未判断 `bookStart` 和 `bookEnd` 的大小关系

    - 未判断 `startTime` 和 `endTime` 的大小关系

- 修改活动(`/api/a/activity/detail:post`)
    - 未判断当前时间导致几个不可修改活动的条件没有实现

- 各需要返回时间戳的请求中，返回的是一个 `datetime` ，而不是一个时间戳
- 数据库不支持 `ASCII` 以外的字符集
- 某些接口中，未判断对应参数是否 *缺失* 直接使用这些参数导致错误
- 某些接口中，未判断参数的长度是否超出数据库中对应域的 `max_length` 导致在某个版本的数据库下错误

人工测试测试出的主要问题

- 上传图像(`/api/a/image/upload`)中，保存路径问题导致无法正确显示图像

并发性测试结果

最大并发数大致为 `100` 左右

测试结果

最终通过了所有测试

系统部署

部署策略

本系统采用 `Docker-compose` 进行部署，选用这种技术主要是因为，首先使用 `Docker` 技术能够很大程度上减少部署时环境配置的麻烦；其次 `Docker-compose` 技术可以同时管理多个容器，而本项目涉及到的容器包括 `Nginx` ， `Web Server` ， `Mysql` ，因此使用该项技术可以方便管理多个容器。

部署方式

首先将 `configs.example.json` 复制一份为 `configs.json` ，修改其中多项：将SECRET_KEY设置为任意想要的长度为50的字符串，将 `Debug` 设置为 `false` （如果需要部署于生产环境），
将 `IGNORE_WECHAT_SIGNATURE` 设置为 `false` ，将 `WECHAT_TOKEN` 设置为随机的字符串，
将 `WECHAT_APPID` 与 `WECHAT_SECRET` 设置为微信官网提供的值，将 `DB_NAME, DB_PASS` 自定为需要 `Docker` 创建的数据库的名称和 `root` 用户的密码，将 `DB_HOST` 修改为 `db` （这个是 `docker-compose.yml` 定义的数据库的 `Host` ，如果需要修改请一并修改 `docker-compose.yml` ），将 `SITE_DOMAIN` 修改为部署服务器的域名；

接着将 `secret_example.env` 复制一份为 `.env` ，将 `MYSQL_DATABASE` 修改为刚刚写

入 `configs.json` 的数据库名，将 `MYSQL_ROOT_PASSWORD` 修改为对应的 `root` 用户密码，将 `ADMIN` 设置为后台管理员账号，将 `ADMIN_EMAIL` 设置为后台管理员邮箱，将 `ADMIN_PASSWORD` 设置为后台管理员密码；

最后使用 `docker-compose up` 拉起服务，加上 `-d` 命令以在后台运行，使用 `docker-compose logs -f -t` 查看日志。

过程管理

持续集成

本组在进行开发的过程中使用 `Github` 做代码管理，使用 `Travis CI` 做持续集成。进行具体的分工之后，组员分别在各自的分支中编写代码，负责编写测试用例的组员先于开发API的组员写出初步的测试用例，推送到远端以进行持续集成。在大的分支合并中，例如完成某个迭代周期的任务后，在合并到 `master` 分支上之前先发起 `Pull Request` ，通过后才进行合并。

迭代过程及相应分工

本项目一共分为两个迭代周期，第一个迭代周期持续两天，总体任务为完成项目开发文档中规定的API，完成后台管理功能相关的API，并进行单元测试与功能性测试，具体的分工为：王泽宇、张佳麟完成相关API，李仁杰、洪方舟编写相关测试用例；

第二个迭代周期持续两天，总体任务为完成**没有在开发文档中说明**的微信公众号所需的抢票等 `Handler` ，进行单元测试与功能性测试，并进行最后的部署，具体的分工为：王泽宇、张佳麟完成相关API，李仁杰编写相关测试用例，洪方舟进行项目部署。