**项目总结报告**

日期：2021/6/25

|  |  |  |  |
| --- | --- | --- | --- |
| 组号 | 第十五组 | 项目名称 | 交我唱 |
| 编程语言 | Java,JavaScript,C,Python | 开发平台和框架 | ReactNative,Spring Boot,Flask,Pytorch |

|  |  |
| --- | --- |
| **项目工作小结** | |
| 1.是否实现了项目立项时的所有需求？列出实现的新增需求和未实现的需求。  实现了大部分需求，取消了对用户体验改善不高但逻辑和实现相当复杂的好友发送消息功能，改为设计更加简洁的点赞加评论功能；考虑到交我唱是一个面向全体大学生的社交K歌软件，暂未加入单点jaccount登陆功能。  我们新增了上述的点赞、评论功能，在第二次迭代的基础上，新增了一系列非功能需求的优化，如要求伴奏的深度学习推理响应时间在用户可接受范围内；APP在不同的机型上都可以流畅运行；单元测试要求正确率100%，100%覆盖核心代码用例；前端系统测试要求黑盒测试为主，全面、详尽地反应APP的每一个功能、每一个按键、尽可能对事件流全覆盖。   1. 采用哪种架构风格？哪些设计模式？   交我唱APP在架构设计时，采用了三层架构（3 Tires Application），即为了“高内聚，低耦合”的思想进行架构设计。  首先是客户层，客户层为了两类客户端，一个是用户所使用的安卓APP客户端，另一个是管理员使用的Web网站管理客户端。客户层通过http请求和基于http的XML请求与服务器（应用层）进行通讯。其间所设计的负载均衡层，由于概念还不清楚，故暂时未画出。  应用控制层接收到http请求后，通过Servlet构件路由到对应的业务组件中。对于Java原生实现的服务逻辑，业务逻辑层调用基础服务层的服务直接进行处理。对于需要Python辅助实现的算法服务，使用http请求与Flask服务器进行请求。  在数据层中，无论是Java原生实现的服务，还是Flask实现的Python服务，会通过数据访问层，通过对应的接口，访问到数据库对象，对数据增删改查。  设计模式我们主要采用了外观模式和适配器模式，外观模式定义的门面是访问子系统所必经的一个入口，门面的设计使得我们的接口能得到统一的处理和转发，使得接口更统一，对前后端进行了解耦。适配器模式主要是针对我们调用的外部系统，如网易云API以及我们复现的一些论文的接口，我们需要封装成适合我们系统内部调用的格式。   1. 技术方案有哪些亮点？   (1)我们利用已有的开源网易云API，实现了在线搜歌功能，这个功能是我们所有在线业务的基础，我们利用私有服务器对查询进行缓存，进一步增加了用户查询的效率。  (2)我们复现了两篇音频领域的学术前沿论文[1,2,3,4]，分别负责我们模块的人声消除算法和歌曲打分算法。我们针对我们的业务场景，对其中的一些魔法参数进行了调整，使得深度学习模型的效果更好。同时把原先的三分钟人声消除算法优化到了数十秒，提升了用户体验。  (3)我们将深度学习模型封装为服务接口，使得后端Spring Boot的和深度学习模型解耦，同时便于在未来的迭代中将我们的深度学习模型进一步封装成微服务，便于动态伸缩来自适应业务请求的变化。  4.是否做了单元测试？是否做了系统功能测试？是否做了性能测试？是否做了兼容性等其他非功能测试？  (1)做了单元测试，核心主类单元测试代码覆盖率达100%，正确率为100%；  (2)做了系统功能测试，针对软件6大核心用例，设计了上百个测试用例进行测试，功能符合预期要求，组内验收达标；  (3)做了性能测试，登陆请求在100并发下响应时间在700ms~800ms左右，请求动态推荐在100并发下响应时间在2000ms~2700ms左右；  (4)做了兼容性测试，软件在华为Mate30，华为P30，华为Nova 5，redmi k20pro机型上现行安卓系统上均可流畅运行，没有发现兼容性错误。 | |
| **项目组成员对项目的贡献度（%）** | |
| 鲍辰：21.6%  梁晨：19.8%  朱家琛：17.8%  郑博洋：20.2%  乐辰阳：20.6% | |
| **软件度量** | |
| 软件代码行数（不包括注解行、空行和复用代码）： | 96613 |
| 复用他人代码行数： | 约9万行 |
| 类的个数： | 50 |

|  |
| --- |
| **经验、教训和建议** |
| 鲍辰： 经验：  进度风险事关全局，哪怕对于这样一个确定性的题目，我们的需求也经常变更导致一系列的返工代价，所以在任何时刻进度风险都是至关重要的，需要能够把控全局合理地为可能存在的需求变更提供一些时间和进度的余量。  教训：  在我们的项目中，我们变更过后端的框架和在不同的服务器之间做迁移，这是因为我们在一开始没有仔细地考虑，并且知识体系不够全面，导致出现了一些盲目的选择。  建议：  虽然我们在项目中已经非常重视需求工程了，但是需求的变更总是非常快速的，有时候vision文档并不能跟上需求变更的速度，这时候就需要我们在一开始需求的时候就好好确立需求基线，及时同步变更。 乐辰阳： 经验：  经过这次的项目，我感觉自己收获了许多。首先是经验方面，主要收获的是编码方面的经验。这里包括了尝试了多种之前未尝试过的编程语言，编程框架，如java，react native等，编译并且整合了不少有名的第三方库，比如ffmpeg。通过运用JNI感受了数据在java层和原生层传递过程，也为运用了原生层代码提高了音频效率而感到了高兴。由于代码量巨大，bug层出，也让我在调试过程中养成了良好的代码风格和命名规范。  教训：  当然这次项目也踩了不少坑，其中最大的教训就是，不要轻易踏进需要许多专业知识的领域。在数字音频处理领域涉及了太多的专业知识，以我有限的时间和自学能力只能做出一些比较naïve的处理工作，实际效果也一般般。这类专业的事情还是必须专业的人来干。第二点教训就是要尽可能多地提交备份自己地代码，为每个版本记录有意义地信息，方便出故障时迅速倒退到正确版本。这点在调试过程中是非常重要的。最后就是除了在编码上花费大量地时间外，更加重要地是明确自己的需求，不然就会想我一样，一会做一点这个一会那个，最后导致了很多的返工。  建议：  所以我的建议是，要有明确的计划，要有良好的编码规范，还要对自己的任务做好更多的了解工作，以对任务难度与风险有更好的预估。 梁晨： 经验：  1.封装一些脚本能有利于自己快速执行一套指令，也能为组员提供一个方便的接口，不需要搞懂流程也能快速实现功能。  2.对于已经调试通过的代码，要妥善备份保存，即分为稳定版和测试版，使得后续更改后出现难以解决的错误，也能版本回退，不至于很长一段时间用不了或者推倒重写。  3.跟别人对接的时候，保存好该版本，也就只需要进行一次对接，后续都能进行成功地复用。  教训：  1.在调通信时，前后端同学要保持良好的沟通，协调好传输方式、传输格式、因调试临时下线等信息，避免因为信息滞后出现错误，然后浪费大量排查错误的时间。  2.选取所用的技术栈时，不能盲目随大流，应当分析清楚自己的功能需求和进度安排，充分考虑功能上的完备性和进度上的可行性后，再根据不同技术方案的特性进行选择。不然容易造成大量返工。  3.要及时回看之前计划的需求设计文档，准确评估进度风险，将进度风险大的功能任务尽量提前安排。  建议：  1.异常处理非常重要，有利于整个软件稳定长久的运行，出现错误后也能更快地找到问题原因进行修正。  2.当碰到一些一时无法解决的报错时，不要跟一个环节死磕，需要积极寻找替代方案，使得后续的任务能够进行下去，计划不被耽搁。  3.当碰到问题时，应当及时进行组内沟通，如果是原计划出现了问题，就应当及时进行修正或更改，才能不耽误进度；否则若是害怕返工，一条路走到黑，会造成更大的损失。 朱家琛： 经验：  1.要学会寻找自学的材料。例如，我在一开始学习appium时，要学习的东西太多导致我无从下手。因为从没接触过jdk，安卓sdk，nodejs等等超多软件。所以一开始出现了各种版本问题与bug。因此，我认为需要找好教程，不要着急上手，要在阅读多篇文档，形成一个大体的架构后，再上手安装软件。  教训：  1.凡事都要早做。因为在一个项目里，你根本不知道接下来要改哪些问题，哪个地方会出bug。因此，最好的方法是，尽早做完尽早对接。  2.写代码一定要上手写，不要看一大堆网课忽略了手上的功夫。因为你会忘记，而且你也不需要这么多知识，只要择其善者从之就好。以功能为导向，就可以很快实现目标。  建议：  1.不要怕，不要眼高手低，要脚踏实地。  2.要经常学会备份自己的代码，不要改了就不管他了。 郑博洋： 经验：  前端开发的过程中获取的经验：前端的代码需要大量的复用设计，这一点需要在开发之初就纳入首要的考虑项目。如果能够在开发之初就搭建合理的框架结构，为组件准备可供重复使用的参数调用方式。可以有效提高代码复用率，从而提高代码整体的灵活性，降低代码结构耦合，提高开发效率。  教训：  前端开发过程中不仅仅需要投入时间在前端上，前后端合作联调也占了很大一部分时间。因此与组员之间沟通就非常重要，如果不能建立高效的沟通协作方式，将极大拖累开发进度。  建议：  软件需求的波动对前端开发的影响非常显著，因此建议在开发初期对需求尽可能细化评估。在新增需求时也及早提上日程。 |

项目组各成员签字：