

项目内容简介

项目开发背景

- 当前，ChatGPT 已经发布并且对公众开放了服务接口，这无疑标志着一个人工智能的新纪元已然到来。通过 ChatGPT 的强势赋能，使得许多传统 workflows 都得到了极大的颠覆与创新，在达到更高效率的同时也能够确保质量。本项目正是在这一背景下，基于 ChatGPT 的公开 API 接口，搭建一个能够根据用户的自然语言描述需求，自动生成 Markdown 格式文档，然后通过 Markdown 解析器处理文件文本内容从而生成 ppt 的软件系统，从而能够在人们的实际文档设计与编写工作中，以一个可靠软件助手的姿态提供辅助，有效提升人们的工作效率。

项目核心功能需求

- ① 在线 PPT 文档编辑，提供与本地编辑器类似的编辑能力
- ② 通过 ChatGPT 生成 PPT 文档内容
- ③ 文档项目管理，包括项目创建删除、项目文件管理、项目分享克隆等
- ④ 用户账号管理

软件定义时期

甲方参与流程

本小组在 PPT Copilot 软件系统项目开发流程中，作为甲方参与项目，与乙方积极交流、沟通，合作，主要参与了软件生命周期中的以下的项目开展环节：

- 问题定义：本小组作为甲方，在项目开展初期对项目需求进行了详细的定义和规划，对软件的功能、性能需求以及目标用户进行了明确的设定。
- 可行性分析：本小组作为甲方，在项目开展初期对项目的可行性进行了分析，对软件的技术可行性、经济可行性、法律可行性以及进度可行性进行了详细的分析。
- 需求分析：本小组作为甲方，同样在项目开展初期明确了软件应用的场景，用户需求，核心产品痛点和主要抓手等需求内容，随后与乙方进行交流沟通，进一步调整并且完善软件系统需求内容，并且将这些需求清晰地由乙方表达。
- SRS 规格说明文档：本小组作为甲方，将上述的三阶段分析内容整理为了详细的 SRS 文档，从而使得乙方能够更好地理解软件的需求，为软件的开发提供了明确的目标。

软件开发时期

甲方参与流程

- 系统部署：本小组作为甲方，根据乙方提供的部署操作说明，正确在 ECS 上部署并且运行了软件系统，以便后续测试工作的开展。
- 单元测试：本小组作为甲方，在乙方将系统源码发给我们后，简单分析了一下项目模块单元内部的实现代码架构与逻辑，随后主要采用白盒测试的相关覆盖方法进行了单元测试，从而保证了软件系统的基本单元功能的正确性。
- 综合测试：本小组作为甲方，采用等价类划分、场景法等黑盒测试方法，通过 Apifox 和乙方实现的 Web 端界面，合理设计测试用例，对于系统功能进行了比较全面且详细的测试，并且将测试结果向乙方反馈，进一步进行沟通交流，从而保证了软件系统的功能性、可用性、易用性、稳定性等质量属性。
- 测试报告：本小组作为甲方，对于软件系统的测试结果进行了详细的整理和总结，撰写了详细的测试报告，从而使得乙方能够更好地理解软件系统的测试情况，为软件的上线提供了明确的目标。

软件维护时期

甲方参与流程

- 系统维护：TODO
- 系统重构：TODO

主功能模块

修改内容

本小组在需求分析阶段，与乙方积极沟通，对于如下 SRS 需求文档中的主功能模块进行了修改：

- 账号管理：简化了账号管理模块功能，移除了用户充值、用户权限管理等部分用例
- 用户充值：完全移除了用户充值功能模块需求，将用户充值功能作为后续的扩展功能进行考虑
- 内容自动化生成：简化了 ChatGPT 对接逻辑，删除了关于 Markdown 中间件生成的用例，由乙方在详细设计中自行决定如何通过 ChatGPT 驱动 PPT 文档内容生成
- 项目管理：删除了部分项目管理用例，使得项目管理模块更加简洁

其它需求

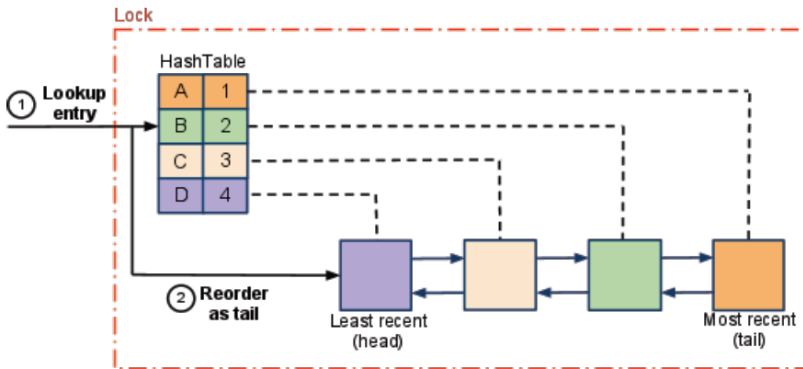
修改内容

- 账号管理：简化了账号管理模块功能，移除了用户充值、用户权限管理等部分用例
- 用户充值：完全移除了用户充值功能模块需求，将用户充值功能作为后续的扩展功能进行考虑
- 内容自动化生成：简化了 ChatGPT 对接逻辑，删除了关于 Markdown 中间件生成的用例，由乙方在详细设计中自行决定如何通过 ChatGPT 驱动 PPT 文档内容生成
- 项目管理：删除了部分项目管理用例，使得项目管理模块更加简洁

优化 SD 服务缓存池策略

- 在之前的单元测试和集成测试工作中，我们小组发现 SD 绘图的最大开销来源于从硬盘加载用户模型这一过程。因此在迭代三中，我们小组的 SD 服务维护人员重新设计了模型加载缓存池算法，采用 LRU 机制，在服务器缓存中维护若干之前用户的模型加载数据结构，并且在缓存池满且缓存未命中时，再从硬盘中将新模型加载到缓存，同时换出 LRU 队列中最近被加载频度最低的模型数据结构，从而有效提高了对于用户绘图请求的响应速度。在此基础上，我们小组的 SD 服务维护人员在经过反复的系统测试和实验后，最终将缓存池大小进行了合理设置，从而使得其既能够保证缓存池中的模型可以满足大部分的用户请求场景，同时也不会因为缓存池过大而导致内存占用过高，影响 SD 服务的运行性能。

优化 SD 服务缓存池策略



Redis 中间件缓存机制实现

- 在迭代 3 中，我们小组在 Tornado 服务和 MySQL 服务之间，架设了 Redis 中间件，采用旁路缓存策略，在数据库事务到来时，首先在 Redis 中查询缓存，缓存未命中时再访问数据库，然后将数据库中的数据读出并且写入 Redis 缓存，从而有效减少了对于 MySQL 数据库的访问次数，提高了系统的运行性能。
- 同时，为了保证 MySQL 和 Redis 的数据一致性，我们小组在迭代 3 会议中经过讨论和架构设计，最终决定采用缓存延迟双删的策略，即在数据库事务执行前先删除缓存，然后在事务执行完毕后，延迟一段时间后再再次删除 Redis 缓存。这样在一般的并发情境下，基本上在数据库服务中不会出现脏数据。

生成图片合法性保证与检测

- 在迭代 2 汇报与甲方的沟通中，甲方提出了一个对于生成图片合法性进行检验的项目需求。因此，我们小组的开发人员经过会议讨论，最终对于这一合法性检验需求，提出了采用 Trie 进行非法 prompt 字段匹配校验，从而过滤非法生成字符，并且在用户 prompt 中，引入 prompt prefix 装饰器机制，对于 C 端调用的 prompt 接口方法进行封装，从而控制 OpenCLIP 编码器产生的文本向量，在每个 step 的收敛过程中都能够保证生成的文本向量在 prompt prefix 的约束下进行的整套合法性检验解决方案，在最终的系统测试中，该方案能够有效保证生成图片的合法性，并且极大地提升了用户裸文本（raw text）输入情况下的生成图片质量。

前端交互逻辑优化

迭代 3 开发工作中，我们小组负责前端开发的同学针对之前迭代 2 中前端交互逻辑的不足，对于前端的交互逻辑进行了优化，主要包括以下几个方面：

- 搜索界面中，支持了通过单击回车键进行搜索的交互逻辑。
- AI 画图界面中，对于 prompt 文本进行了更加详细的描述，加入了提示性文本引导用户操作，避免用户使用时因为功能和参数过于复杂导致的困扰
- 修复了登录界面中，登录栏与注册栏在浏览器页面缩放下的错位问题。
- 修复了若干界面显示样式问题，使得整个系统的前端交互逻辑更加合理。

单元测试

对于软件系统的单元测试，我们小组分别采用了使用 Apifox 工具和 Python requests 库请求的方式，请求每个实现接口对应的 url，并且分别验证接口返回数据格式的正确性、接口根据实际业务请求内容做出响应的正确性和接口在高并发情景下的执行正确性。首先，Apifox 接口测试流程大致如下。在每个接口定义中，填入参数并且发送，比如这里以 /user/register 接口为例：



单元测试

Apifox 报告

测试场景 用户管理接口压力测试
运行时间 2023-05-22 11:26:43
运行工具 Apifox v2.2.39

	总数	失败数
循环数	50	0
HTTP 接口请求数	300	0
断言数	0	0

总耗时 1m 52.80s
总返回数据 27.45KB
平均接口请求耗时 70ms

通过率 100.00%
失败率 0.00%
未测率 0.00%

集成测试

随后，在对于所有的功能接口进行单元测试正确无误后，我同样使用 Apifox 测试工具，对于系统进行集成测试。所谓集成测试，是在上文进行的单元测试基础上，将所有模块按照设计要求（如根据结构图）组装成为子系统或系统，并且以子系统为单位进行测试工作，从而验证每个模块在经过组装后，是否依然能够正确运行。这里，我同样以用户管理功能模块中的用户注册、登录和退出登录为例，对于我进行的集成测试工作具体方法和流程进行阐述。首先，在 Apifox 中设计一套模拟新用户访问系统用户管理模块的具体操作逻辑，如下所示：

集成测试

用户注册、登录、退出登录

删除

编辑

目录

创建者 LiQiTong

优先级 P0

创建时间 5 分钟前

修改时间 几秒前

修改者 LiQiTong

测试步骤

测试数据

测试报告

✓ 已选 5 项 移除

≡

* 运行环境

lo localhost:8...

≡

测试数据

停用 ?

循环次数

5

?

遇到错误时

忽略

?

线程数

10

?

间隔停顿

0

毫秒 ?

✓ 保存请求/响应详情 ?

全部请求

▼

保存变量变化值 ?

使用全局 Cookie

保存 Cookie 到全局 ?

↺ ↻ ↻

✓ 循环10次

✓ POST 用户注册 (用户注册)

✓ POST 用户登录 (用户登录)

✓ If session_key 等于 null

✓ POST 用户退出登录 (用户退出登录)

拖入或 添加步骤

拖入或 添加步骤

➕ 添加步骤

集成测试

可见这里模拟了一个典型的用户访问软件系统交互逻辑，即注册新账户后登录，如果成功则储存 session_Key，访问系统其它服务，最后登出账户，在服务端的 Redis 缓存中相应移除用户 session 标识，并且更新用户 stat 字段为离线。该测试用例运行结果如下：

运行用例

X

用户注册、登录、退出登录



全部 成功 失败

系统测试

最后，在上述的两个测试阶段工作全部完成，并且测试结果验证正确无误后，我们小组的测试人员从软件系统客户端用户的角度，将实现的软件系统客户端打包发布，邀请同学和相关非开发人员的第三方用户来使用软件系统 C 端页面并且给出评价。同时，我们小组测试人员也对于其中 C 端的相关主要功能模块，从系统的层面进行了一个比较完整的用户交互逻辑测试，并且观察客户端的数据响应情况，从而能够有效地评估软件系统作为一个整体的性能表现。

目标和预期

- 围绕甲方 SRS 需求文档为核心，高质量地完成相关功能需求的开发任务
- 围绕功能用例，时刻开展自动化测试，保证项目的稳定性和可用性
- 上线项目，邀请用户体验并给出测试反馈，收集用户意见和 Bug 反馈，为第三次迭代的内容规划提供参考

信息和工具

- 小组中对于相关功能需求分析实现，很大程度上参考了甲方提供的 SRS 需求文档，以及甲方提供的用例图，从而使得设计与实现内容与甲方需求最大程度地做到匹配
- 技术上，我们小组借助相关网络资料以及以往的开发经验，并且和相关从业专业人士进行沟通，充分对于当前行业的流行技术生态进行调研，选择最有利于敏捷开发的技术栈，从而使得项目开发效率得到了很大的提升

困难与阻碍

- 我们小组开发在迭代三遇到的最大的困难是在项目部署到服务器运行并测试时，遇到项目在实际的网络环境与本地环境下运行的差异，导致项目之前的许多功能实现可能在本地运行高效、良好，但是在网络环境下则由于网络带宽、高并发场景等原因，导致项目运行效率低下。这导致我们虽然很快地实现了相关的项目需求，但是依然需要负责相关功能模块的同学花费许多的时间对于原本功能实现的底层算法与代码逻辑进行优化，尽可能地降低算法对于系统资源的消耗。这也使得我们深刻意识到了实际生产环境下的项目开发与测试的重要性，以及对于项目的高效性、可用性的重视程度，让我们对于实际可用的工业化项目的敏捷开发流程有了更加深刻的认识。

优势和创新

- 得益于我们小组在迭代 1 中搭建的良好项目管理架构，我们在迭代三中进一步对于整个管理流程进行了完善，高度规范化了组员从功能开发、自动化测试、代码提交、审查、合并、部署、上线等整个项目开发流程，从而使得我们小组的开发人员能够更加专注于项目的功能实现，而不是处理各种由于项目流程管理不当产生的问题，使得我们小组的开发效率得到了进一步的提升。
- 我们小组在本次迭代中，着重进行了后端服务的相关优化升级，对于相关的服务代码架构进行了重构与测试，成功在高并发的情景下，保证了项目的高效稳定运行，使得我们小组的项目更加具有可用性和可靠性。

结果和进度

- 我们小组在每周的组会上，都会各自汇报本周自主开发以及集中开发中完成任务的情况，并且由会议记录人员进行统计。最终结果表明，小组在本次迭代中，完成了甲方提出的所有功能需求，并且在自动化测试方面也取得了很大的进展，使得我们小组的项目更加稳定可靠，结果还是比较让我们感到欣喜和自豪的。

情绪状态

- 我们小组成员在本次迭代开发过程中，都非常重视开发工作，花费了大量时间完成相关工作项。期间许多组员感染新冠阳性，但依然带病线上参与开发工作，讨论项目进度，完成相关任务，体现了我们小组成员的责任心和敬业精神，也使得我们小组的项目开发进度得到了保证。

心得体会与开发意义

- 通过本次迭代任务，我们小组成员之间的配合更加融洽并且具有默契，能够作为一个团队更加高效与良好地完成相关的项目开发任务，并且更加熟悉了类似的大型项目的开发流程与管理模式，为之后的下一次迭代乃至今后学习工作中的类似项目开发情景打下了良好的基础。

参考文献

