

# 软件工程（实验班）项目结题报告

## 金桥通

张佳豪 2200013093

汪之立 2200012966

## 一、项目概述

### （一）项目目的

“金桥通”是一个集成现代化智能技术手段的一个安全、高效、便捷的 P2P 借贷平台。它运用先进的信用评估系统，结合大数据分析和人工智能技术，为用户提供个性化的贷款和投资服务。平台旨在打破传统金融服务的局限，为中小企业和个人用户提供更灵活的融资渠道，同时为出借人带来稳健的投资回报，共同构建信任 and 价值的金融桥梁。

### （二）项目背景

该项目致力于打通个体小额借贷需求，拓展平民金融市场，从而在一定程度上提升社会的消费能力。在社会消费欲望日渐低迷的当下，借贷双方在达成交易方面出现了许多困难。而“金桥通”依托于人工智能领域的最新进展，将大数据、机器学习和通用大语言模型融合进传统的 P2P 平台从而为用户提供更加精准的服务。一方面，自由的借贷市场可以将金融服务推向社会的最深处；另一方面，该项目采用的人工智能技术可以增强借贷双方的信心，并提供合适的引导，真正成为借贷双方之间的可靠桥梁。

### （三）项目技术概览

“金桥通”面向的用户群体主要是有投资或者贷款需求的个人，因此项目客户端部署于 Android 平台，从而对广大用户提供更为方便的金融服务。本项目采取服务器-客户端的分离设计，从而增强用户信息的安全性，并提高降低了单位访问的服务成本。

服务端基础框架采用了 Rust 语言并基于 TCP 协议和高性能网络库进行搭建，实现了本地会话缓存、自动定时清理、可拓展的定制服务等底层内容。服务端数据方面依托于 MongoDB 数据库实现数据的安全分离，服务端模型方面采用本地预训练的 python 机器学习模型实现用户评分的快速预测。另外，服务端还借助如 STMP、OpenAI API 等第三方支持来为用户提供多样化的可靠服务。

客户端框架采用了 React Native 框架并基于 TCP 协议搭建了可拓展的通用传输层。同时客户端实现了选择性缓存机制来节省单用户带宽。此外，客户端搭建了简洁易用的 UI 界面来改善用户体验。

项目的开发中采用“语雀”作为项目文件的集中管理平台，同时使用 Github 作为代码托管和协作平台。开发全程均采用 ChatGPT4 和 Copilot 进行支持，在 7 天的时间内基本完成全部内容。

### （四）项目规模概览

截至 01/06/24，项目的自主代码规模服务端有 2710 行，客户端有 4811 行。

#### 1、 服务端代码统计：

Language	Files	Blank	Comment	Code
Rust	11	217	40	2611
JSON	2	0	0	54
Python	1	9	2	24
TOML	1	5	1	21
<b>SUM</b>	<b>15</b>	<b>231</b>	<b>43</b>	<b>2710</b>

2、客户端代码统计：

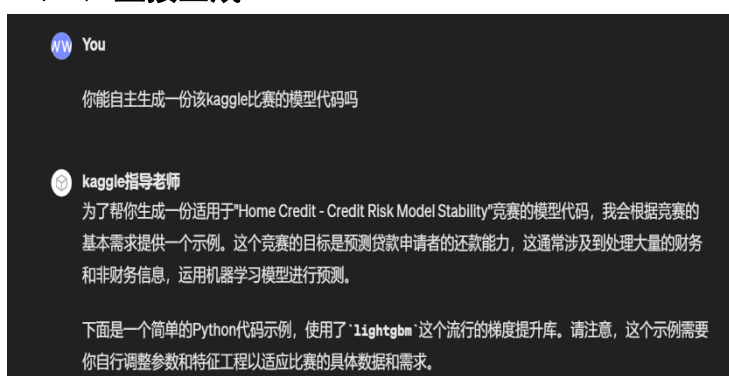
Language	Files	Blank	Comment	Code
JavaScript	37	344	24	4787
JSON	1	0	0	24
<b>SUM</b>	<b>38</b>	<b>344</b>	<b>24</b>	<b>4811</b>

## 二、复现阶段

本项目对 Kaggle 题目“Home Credit – Credit Risk Model Stability”进行复现，并利用其为用户进行金融信用评分。

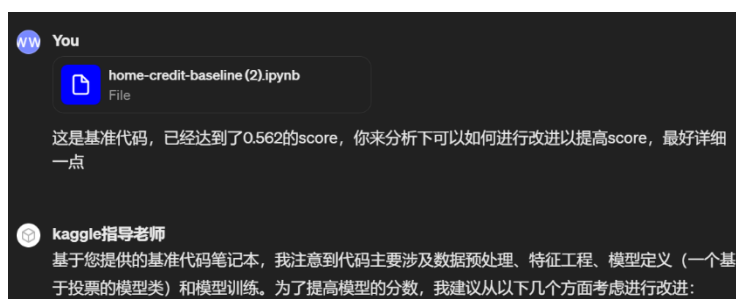
复现中首先使用 ChatGPT4 来尝试针对题目直接生成对应的模型代码，然后再尝试针对 baseline 代码进行针对优化——向 GPT4 呈现 baseline 代码，然后引导它针对优化。

### （一）直接生成



在直接生成的尝试中 ChatGPT4 仅仅生成了一个通用的机器学习样例代码而没有针对该问题特定的数据集和问题类型进行处理，IO 方面出现接口错误，同时也并没有对特征进行处理，无法从零生成一份可以运行的模型代码。

### （二）针对优化



继续引导 ChatGPT4 使用模型融合方法对 baseline 进行优化。

但是依然出现了接口不匹配、显存超过限制等诸多问题。将问题反馈给 ChatGPT4 提示它进行修正，ChatGPT4 通常能正确地解释错误日志但是无法对代码进行正确的修改。

### （三）结果与评价

由于 GPT 无法对本 Kaggle 题目进行自主生成或优化，所以本项目最终采用 baseline 代码模型。

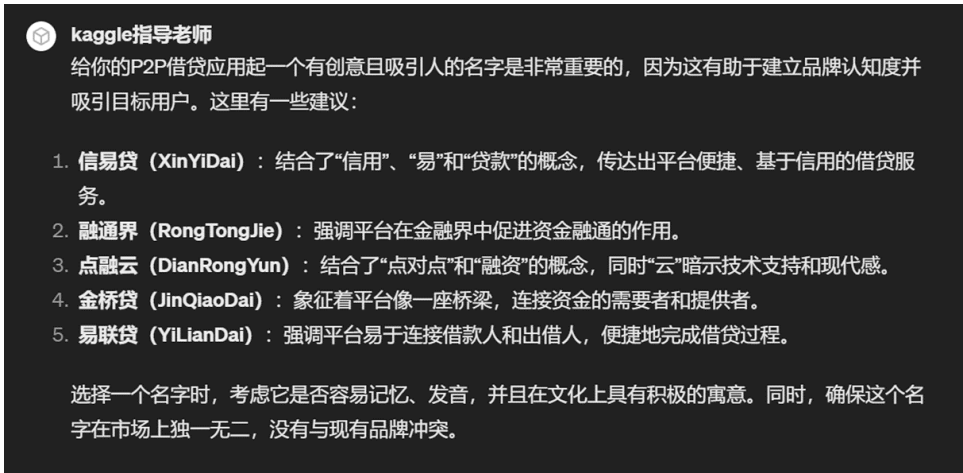
这一阶段的用时约 4 个小时。

大模型在复现阶段的表现不尽如人意，这或许意味着当下大语言模型并没有强大的处理复杂新问题的能力。

### 三、需求分析阶段

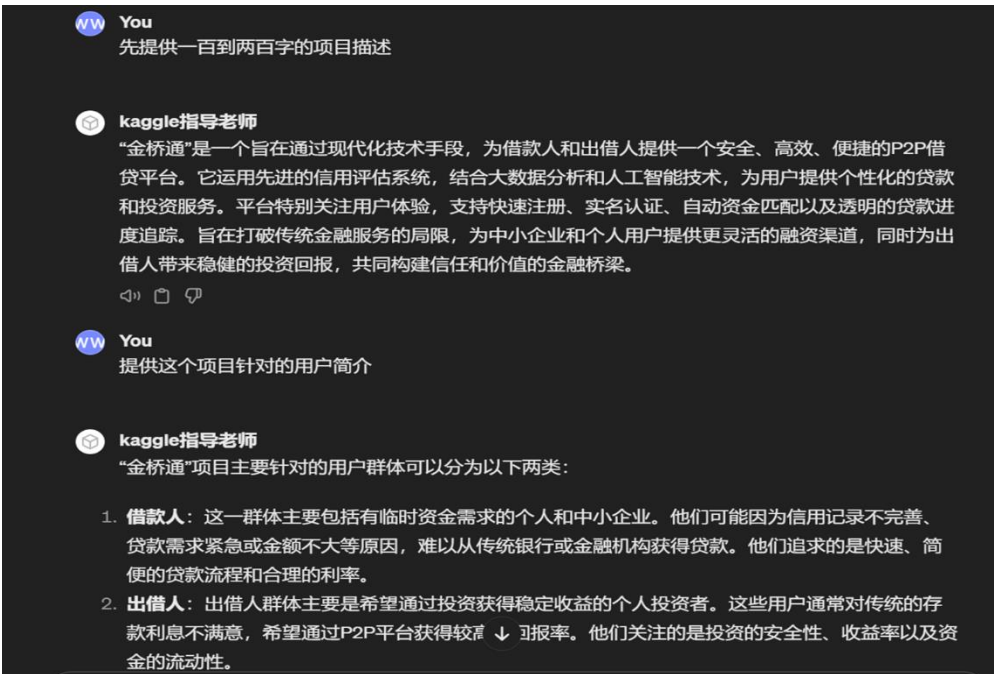
本项目在需求分析阶段主要使用**假想设计**的方法在大语言模型的辅助下进行需求分析并针对需求进行用况设计。

#### （一）确定名称



我们首先为 GPT 提供“金桥通”项目的核心特点“智能模型”、“P2P 借贷平台”，并依据 GPT 的建议确定项目名称为“金桥通”。

#### （二）确认项目特点和目标用户



我们然后在 GPT 的指导下确定项目的基本描述和针对的用户群体。

#### （三）需求分析

在项目基本特点的目标用户的基础上，我们指导 GPT 针对全部用户生成统一需求，再针对不同用户类型生成特色需求，并区分功能性和非功能性需求，并采

用清晰的格式进行列举。

<div><div><div>针对借款者的功能性需求</div><div><div>1. 借款申请处理:</div><div>• 简化的借款申请流程, 包括贷款金额、用途、期限等信息的输入。</div></div><div><div>2. 信用评分与贷款条件:</div><div>• 自动基于用户的信用历史和财务状况提供信用评分, 明确贷款条件。</div></div><div><div>3. 贷款进度追踪:</div><div>• 允许借款者实时查看贷款申请的审批状态和资金到账情况。</div></div><div><div>4. 还款管理:</div><div>• 提供灵活的还款计划和多种还款方式选择。</div></div></div><div><div>针对所有用户的共同功能性需求</div><div><div>1. 用户账户注册与管理:</div><div>• 支持一次性注册, 允许用户根据需要选择身份 (投资者或借款者) 并需重新注册。</div></div><div><div>2. 实名认证与安全:</div><div>• 一次实名认证适用于用户账户的所有活动, 确保平台交易的安全性和合规性。</div></div><div><div>3. 用户资料与隐私保护:</div><div>• 加密用户资料, 保护个人隐私不被未授权访问。</div></div><div><div>4. 客户支持与服务:</div><div>• 提供全面的客户支持服务, 包括FAQ、在线客服和电话支持, 帮助解决用户的疑问和问题。</div></div></div><div>通过这样的功能性需求分析, 我们确保“金桥”能够为用户提供定制化服务, 同时保持平台操作的灵活性和便捷性。这将有助于建立一个高效、安全、用户友好的P2P借贷平台。</div></div>	功能性需求	<div><div>统一需求:</div><div><div>• 用户注册、登录、实名认证</div><div>• 用户财务信息的统计服务</div><div>• 资金中转平台服务</div><div>• 电子化借贷协议公证</div><div>• 大模型财务分析评价</div><div>• 基于特定模型的长期财务追踪分析</div><div>• 专业金融人士财务分析建议服务</div></div></div> <div><div>借款人:</div><div><div>• 贷款申请</div><div>• 出借意愿匹配</div><div>• 信用评分</div><div>• 还款计划管理</div><div>• 贷款状态追踪</div></div></div> <div><div>出借人:</div><div><div>• 发布资金提供和收益意愿</div><div>• 贷款意愿匹配</div><div>• 风险评估、收益预测、对接成功率预测</div><div>• 出借资金状况追踪</div></div></div>

<div><div>可用性</div><div><div>• 用户界面: 应直观易用, 支持多语言, 确保各年龄段用户都能轻松导航。</div><div>• 文档与帮助: 提供详细的用户手册、FAQ和在线帮助文档。</div></div></div> <div><div>性能</div><div><div>• 响应时间: 系统响应时间应在3秒以内, 确保快速响应用户操作。</div><div>• 并发处理: 能够支持至少10,000名用户同时在线操作而不降低性能。</div><div>• 数据处理速度: 贷款申请和审批流程应在24小时内完成。</div></div></div> <div><div>可靠性</div><div><div>• 系统正常运行时间: 确保99.9%的在线时间, 每周定时进行维护和更新。</div><div>• 数据备份: 自动进行每日数据备份, 防止数据丢失。</div></div></div> <div><div>安全性</div><div><div>• 数据加密: 使用SSL加密技术保护数据传输过程中的信息安全。</div><div>• 访问控制: 实现基于角色的访问控制, 确保只有授权用户才能访问敏感信息。</div><div>• 防御机制: 部署防火墙、入侵检测系统 (IDS) 和入侵防御系统 (IPS) 等安全机制, 防止恶意攻击。</div></div></div> <div><div>扩展性</div><div><div>• 模块化设计: 系统采用模块化设计, 便于功能的添加和修改。</div><div>• 技术框架: 选择能够支持高并发和大数据处理的技术框架。</div></div></div>	<div><div>扩展性</div><div><div>• 模块化设计: 系统采用模块化设计, 便于未来功能的添加和修改。</div><div>• 技术框架: 选择能够支持高并发和大数据处理的技术框架。</div></div></div> <div><div>法律和规范遵守</div><div><div>• 合规性: 确保平台的运营符合金融行业法规, 包括反洗钱 (AML) 和客户身份验证 (KYC) 规范。</div><div>• 隐私保护: 遵守相关的数据保护法律, 如GDPR或中国的网络安全法, 保护用户个人信息不被泄露。</div></div></div> <div>通过满足这些非功能性需求, “金桥”项目将能够提供一个安全、可靠、高效和用户友好的P2P借贷平台, 确保长期稳定运营, 并赢得用户的信任 and 满意。</div>
<div><div>非功能性需求</div><div><div>• 数据加密: 使用SSL加密技术保护数据传输过程中的信息安全。</div><div>• 访问控制: 实现基于角色的访问控制, 确保只有授权用户才能访问敏感信息。</div><div>• 防御机制: 部署防火墙、入侵检测系统 (IDS) 和入侵防御系统 (IPS) 等安全机制, 防止恶意攻击。</div></div></div> <div><div>扩展性</div><div><div>• 模块化设计: 系统采用模块化设计, 便于功能的添加和修改。</div><div>• 技术框架: 选择能够支持高并发和大数据处理的技术框架。</div></div></div>	<div><div>安全性:</div><div><div>• 遵守金融行业法律法规</div><div>• 数据加密、数据备份</div><div>• 网络攻击防御机制</div><div>• 隐私保护</div></div></div> <div><div>易用性:</div><div><div>• 直观易用的用户界面</div><div>• 详尽的用户手册和简单易上手的操作引导</div><div>• 响应时间: 一般平台系统响应时间应在 0.5 秒以内</div><div>• 并发处理: 能够支持至少 10,000 名用户同时在线操作</div><div>• 业务处理速度: 贷款申请和审批流程应在 24 小时内完成</div></div></div>

#### (四) 用况设计

在得到项目要解决的各种需求之后, 首先引导 GPT 针对需求生成大量可能的用况类型, 然后人工从中选择部分确定为本项目采用的用况。

使用 GPT 进行逐一生成, 最终得到了完整的用况, 下面列举其中两个

序号	功能	序号	功能	序号	功能	序号	功能
1	用户注册	11	投资回款接收	21	交易记录查询	31	信用评分更新
2	用户登录	12	贷款审批流程	22	投诉与反馈管理	32	平台公告管理





## **（五）结果与评价**

在这一阶段中 GPT 完成了 95%以上的工作，在我们确定项目的基本特点和方向后 GPT 生成了“金桥通”的全部需求和用况。

这一阶段的用时约 8 个小时。

需求分析阶段是大语言模型表现最出色的阶段之一，具体体现为大语言模型可以依据简单的提示给出全面而具体的需求，并依据需求生成合理的用况。其已经基本具备独立完成需求分析和用况设计的能力。



四、面向对象分析（OOA）

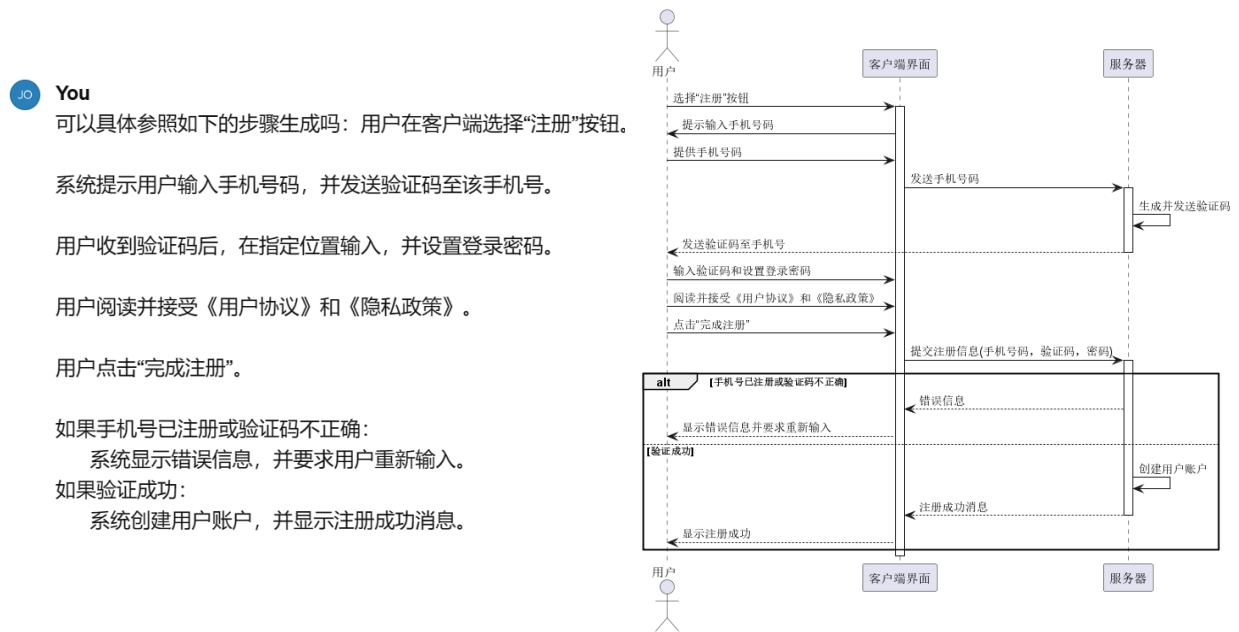
对象	工作
用况	生成了顺序图
服务端的复杂逻辑	生成了状态图
服务端-客户端通信过程	生成了活动图
服务端	生成了类图
服务端-客户端架构	在大模型辅助下设计了简单架构

（一）顺序图

1、初步生成

（1）生成过程

我们生成顺序图的方法是提示大模型使用 UML 语言去格式化用况。



GPT 有能力从中提取参与主体和逻辑关系，并输出为 UML 语言，但结果仍然存在表意不完备和上下不统一的部分。我们人工甄别问题并进行细化提示，大多数问题都可以在提示后被解决

（2）结果与评价

使用该方法生成的顺序图在时序逻辑上的处理**基本是正确的**。用况中的逻辑判断、选择、循环等条件一般可以被清晰地描述。

尽管 GPT 可以生成“大体正确”的针对用况的顺序图，但是这样的顺序图常常过于简单。

我们认为原因可能是 GPT 的输入只有以自然语言描述的用况，而是缺失了具

体的对象或者函数名称。这导致生成的顺序图中往往只有一个笼统的称呼如客户端界面，服务端等，不利于后续的解耦拆分。

对此我们改进方法进行细化的生成。

2、 细化生成

(1) 框架设计

由于只依据自然语言描述的用况 GPT 难以生成具体的顺序图，我们故而考虑提供具体的服务端架构设计以及通信框架，并对格式进行约定。

服务端架构设计

项	描述
客户状态管理	客户状态信息存储在服务器端，由服务端统一管理。
新建连接处理	服务器为每个新连接分配一个线程，负责处理该连接的请求。
资源管理策略	当服务器资源占用达到一定程度时，拒绝新的连接请求，以防止服务器过载。
连接通信方式	客户端与服务器之间采用 TCP/IP socket 连接进行通信。
数据请求处理	服务器接收到客户端请求后，由对应线程向数据库查询数据，并将其返回给客户端。
并发控制机制	采用适当的并发控制机制，例如线程池、信号量等，避免资源竞争和死锁问题。
通信安全保障	采用 TLS 等加密协议对通信数据进行加密，确保数据安全。
异常网络处理	考虑到网络异常情况，当连接断开时，服务线程不立即关闭，而是允许客户端尝试在一定时间内重新连接。
线程管理	服务端采用线程池管理所有的服务线程
状态机设计	每个服务线程都采用状态机模式设计，方便管理和调试。
登录状态存储	客户登录状态、用户资金等敏感信息存储在用户专有会话中，方便访问和处理。
客户端状态管理	客户端状态主要由客户端界面决定，并根据需要向服务器请求刷新数据。
安全性与负载	将不需要安全保护的逻辑尽量放在客户端，减轻服务器负担。
数据传输格式	客户端与服务器之间的数据传输采用 JSON 格式，方便解析和处理。
传输 API 设计	设计统一的传输 API，规范数据传输方式，方便开发和维护。

通信 API 框架（简化）

API 名称	描述	请求 JSON	返回 JSON
注册	用户注册	{ "type": "register", "user": "string", "username": "string", "password": "string", "extra": null }	{ "status": "integer", "request": { "type": "register", "user": "string", "username": "string", "password": "string", "extra": null }, "extra": null }
登录	用户登录	{ "type": "login", "user": "string", "username": "string", "password": "string" }	{ "status": "integer", "request": { "type": "login", "user": "string", "username": "string", "password": "string" }, "extra": null }

		"string", "password": "string", "extra": null}	"extra": null, "user": "string", "extra": null}
获取 信息	获取用 户信息	{"type": "get", "user": "string", "extra": null, "content": [{"name": "string", "time_trace": "string", "extra": null}, ...]}	{"status": "integer", "request": {"type": "get", "user": "string", "extra": null, "content": [{"name": "string", "time_trace": "string", "extra": null}, ...]}, "extra": null, "content": {"name": "string", "status": "number", "extra": null, "content": "A specified thing(or null)"}}
上传 信息	上传用 户信息	{"type": "post", "user": "string", "id": "string", "extra": null, "content": {"Name": "A specified thing", ...}}	{"status": "integer", "request": {"type": "post", "user": "string", "id": "string", "extra": null, "content": {"Name": "A specified thing", ...}}, "extra": null, "content": {"Name1: {"name": "Name1", "status": "number", "extra": null, Name2: ..., ...}}
基本 构成	基本的 API 结构	{"type": "string", "user": "string", "content": "...", "preserved": "string", "extra": null}	{"status": "int (200 for success)", "user": "string", "sessionExpired": "bool", "preserved": "string", "content": {"success": "bool", "error": "string"}, "extra": {"error": "string"} (if exist)}

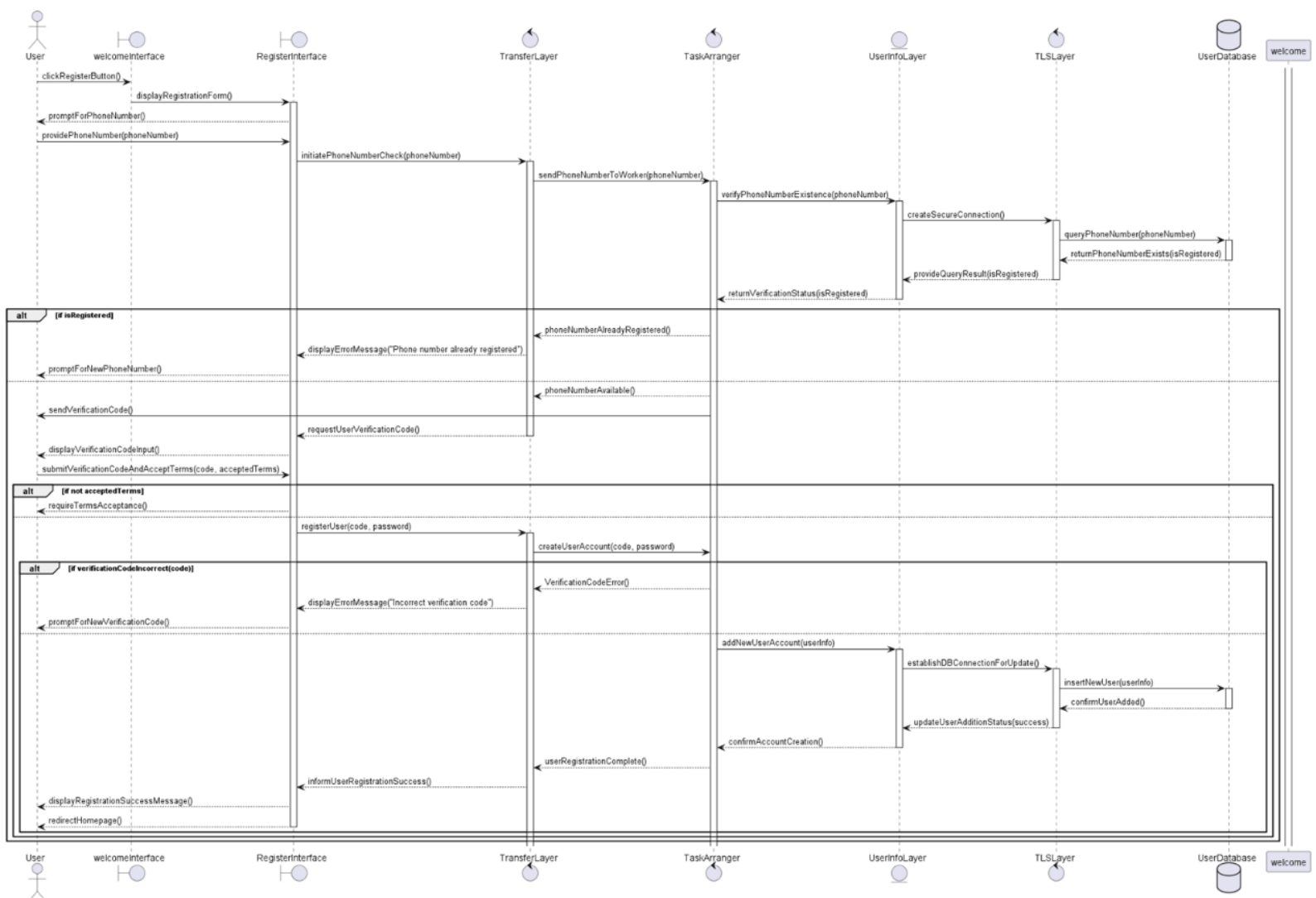
数据格式（部分）

信息项	规范要求
用户名	4-10 个字母/数字
密码	6-12 个字符
头像	base64 编码
邮箱	合法邮箱格式
账户余额	float 64
账户评分	float 64 (0.0~1.0)

以上内容的设计方法主要为在 GPT 以及互联网信息辅助下的人工设计，其中服务端架构由单人设计，通信 API 和数据格式由双人协商设计。

(2) 细化生成过程

我们首先将设计的框架同用况一并发送给 GPT 作为已知信息，其次我们反复强调输出顺序图的格式细节，如函数化过程、规范主体名称等。在此基础上我们对不满意的部分进行人工针对提示，如强调任务分发的过程，给出中间层的具体名称等。以下是一个生成样例。

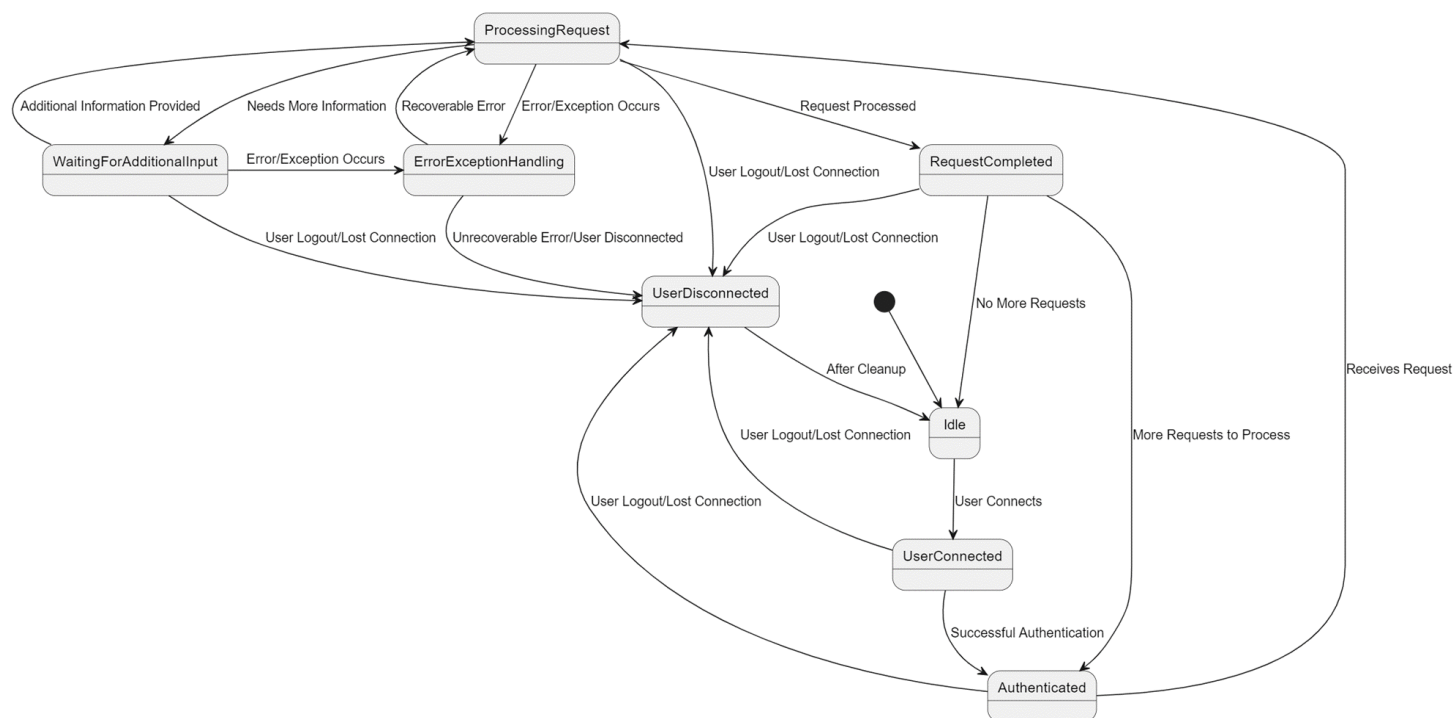


该顺序图十分详细，主体具有统一性，流程也有足够的细分粒度。

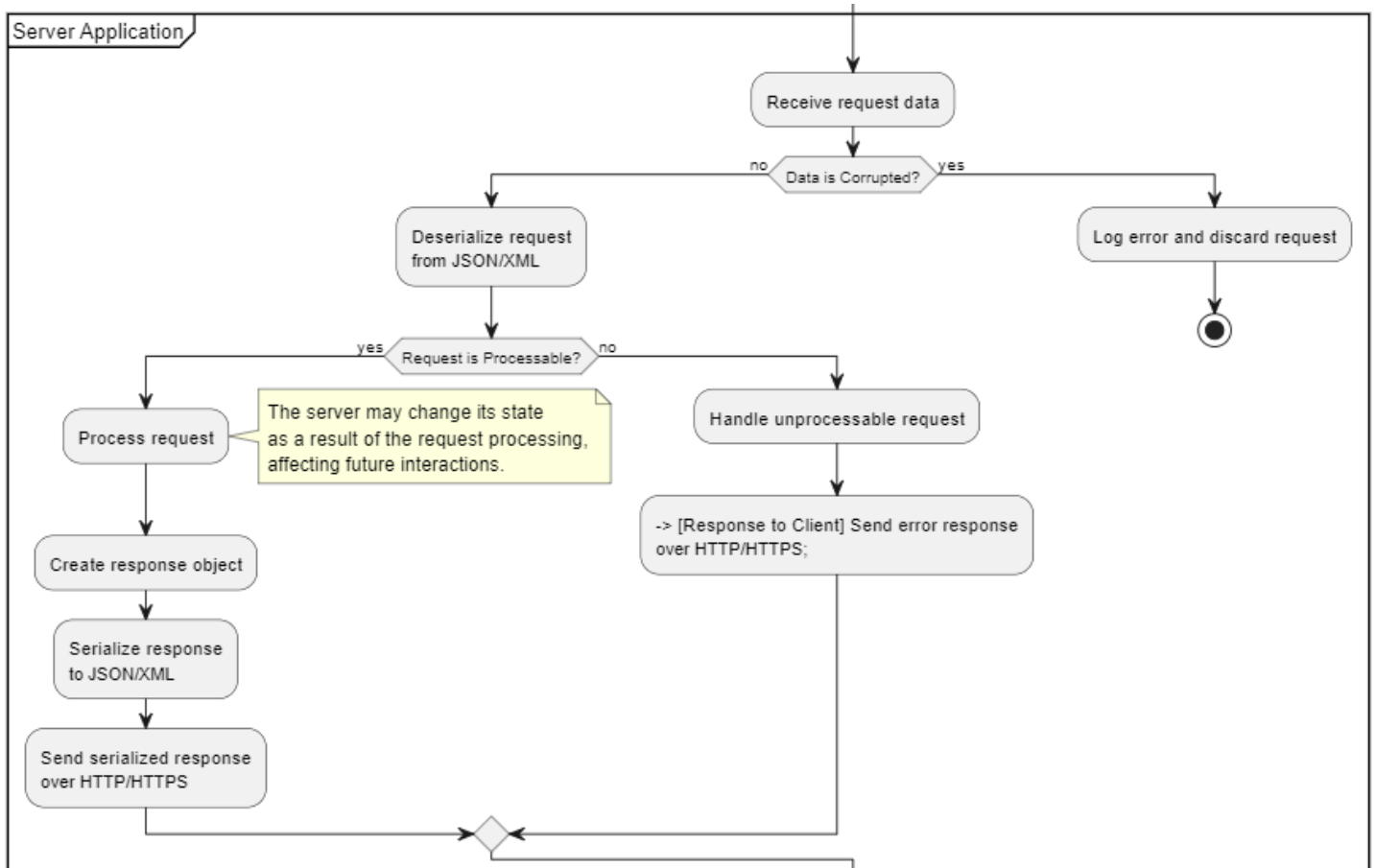
## （二）状态图、活动图 and 类图

状态图、活动图 and 类图的生成遵循设计图的范式：向 GPT 提供框架设计、强调输出规范、人工发现问题并针对提示。

### 服务端服务线程状态图



通信过程活动图（部分）



在上方的例子中，在大模型生成的状态图和活动图有较为良好的表现。

但如果涉及的对象过于复杂，则生成的图不能够达到标准，比如生成有嵌套状态的状态图。此外，由于后续图生成基于的内容量更多，所以 GPT 会时常出现前后主体名称不统一或者凭空捏造内容的情况。有时无中生有的内容在当时的语境也较为正确，单独看生成内容难以发现，必须要前后结合才能发现端倪。

### （三）结果与评价

在大模型的辅助下，我们在较短的时间内完成了项目中复杂流程和对象的分析设计，并产出了逻辑清晰的 UML 图示。大模型辅助下效率提高的部分在于 UML 语言学习成本下降和图的快速生成修改。

此阶段的用时两人均不超过 8 个小时。

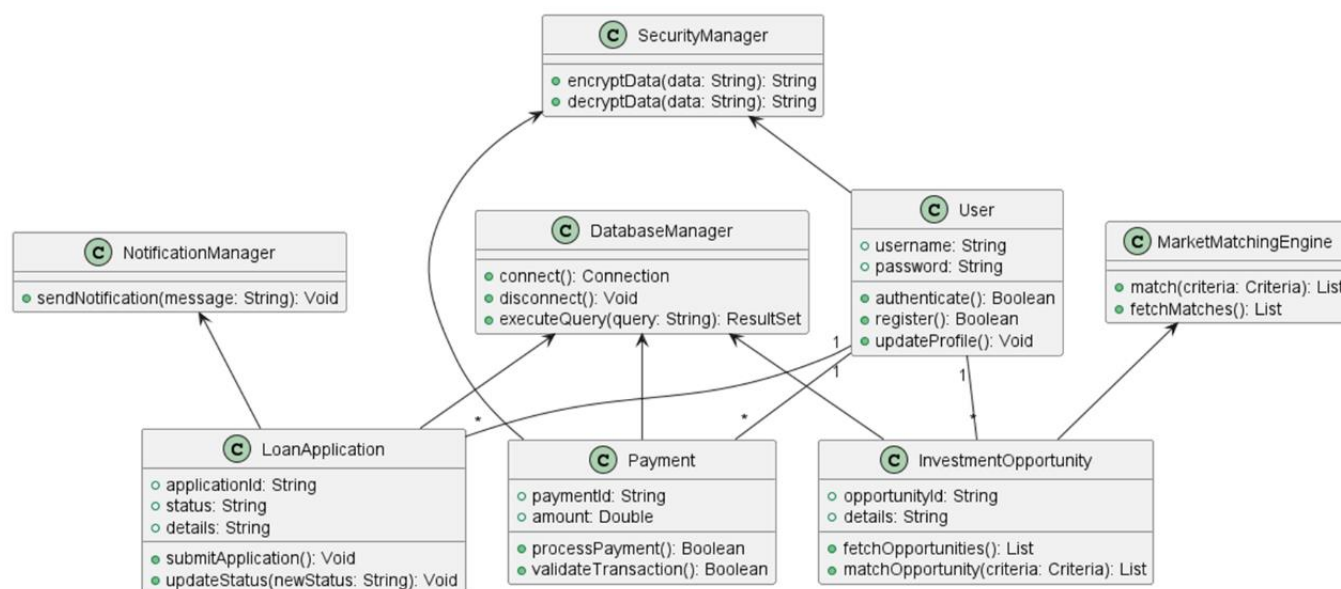
总的来说，模型辅助的 OOA 具有快速的巨大优势，但是容易出现前后不一致和图示过于简单的问题。所以无论是在生成前的提示设计，还是生成后的检查同步，人工都是不可或缺的一环。但是，相比于传统的人工 OOA，在大模型辅助下的 OOA 过程的敏捷性允许我们快速迭代多版设计，在更短时间内达到人工设计的效果。

## 五、面向对象设计（OOD）

### （一）客户端

#### 1、 整体生成

我们首先基于 OOA 中生成的顺序图提示 GPT 生成客户端的总体类图。



但是该类图十分简陋，缺失了许多部分，不能作为客户端的对象设计。

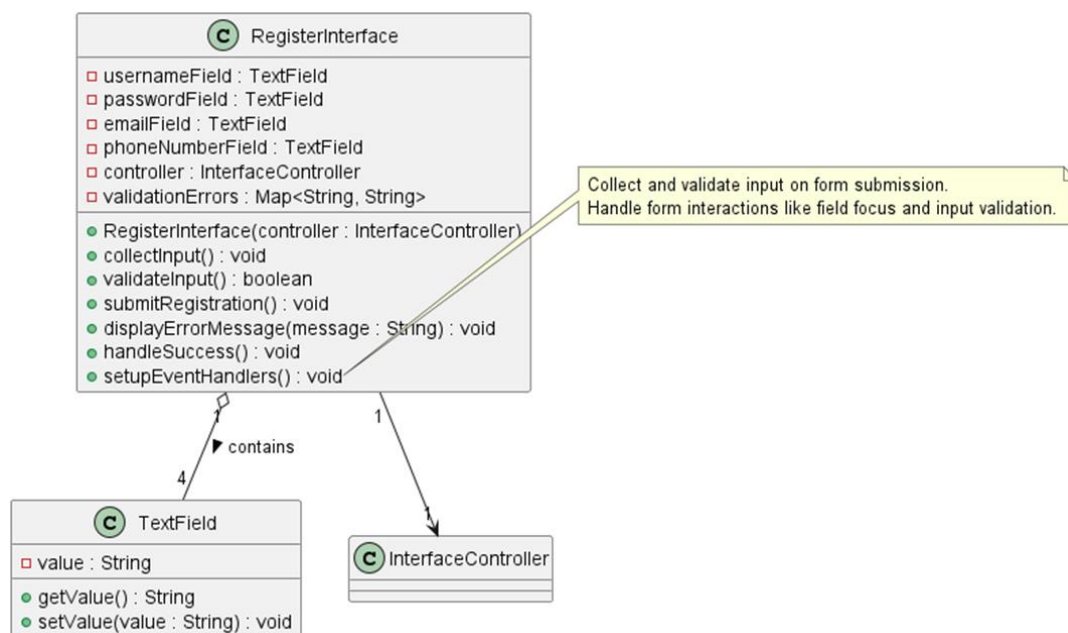
#### 2、 分类生成整合

由于 GPT 无法直接根据顺序图生成客户端的合理类设计，所以我们分别针对每一个用况的顺序图生成相关的类，然后将所有的类整合成客户端的完整类图。

对于某一个具体的顺序图的相关类我们的具体提示策略是：

- 输入某一用况的顺序图
- 强调人机交互部分
- 要求提取出相关的组件，如按钮、文本框等
- 将组件、交互函数提取整合相关的类
- 人工检查，提示修正

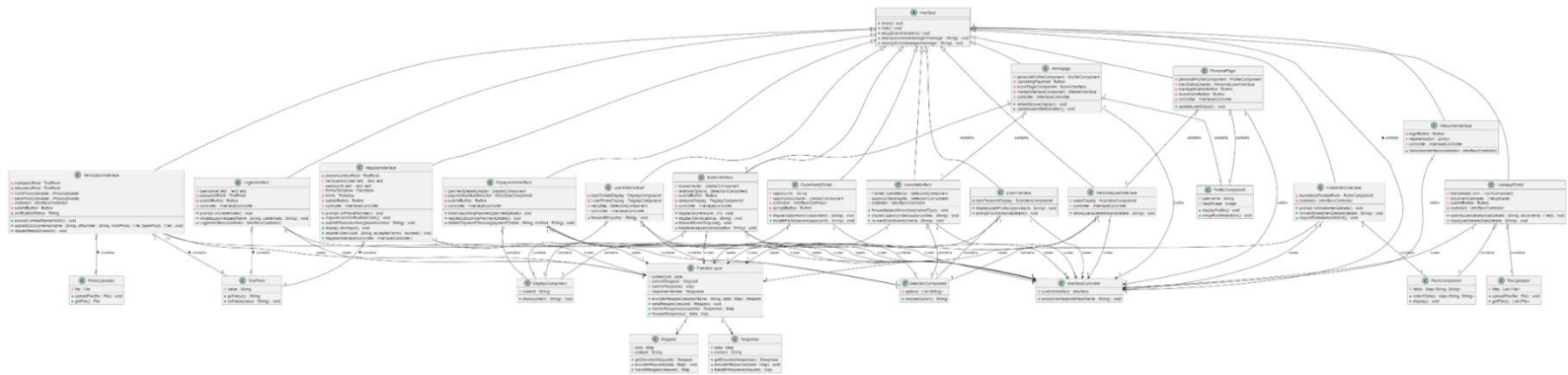
#### 注册界面的人机交互相关类



优点	缺点
确实限制在了人机交互范畴	返回函数仅仅给出类型，没有具体格式
组件、交互函数提取完整	通用组件提取程度不够
能够正确表现类之间的关系	结合前端框架有多余的控制逻辑

实际提示工作除了第一次需要完整提示，之后的往往只需要模仿输入用况就行。最常出现的情况是在图中包括了不属于人机交互的函数。

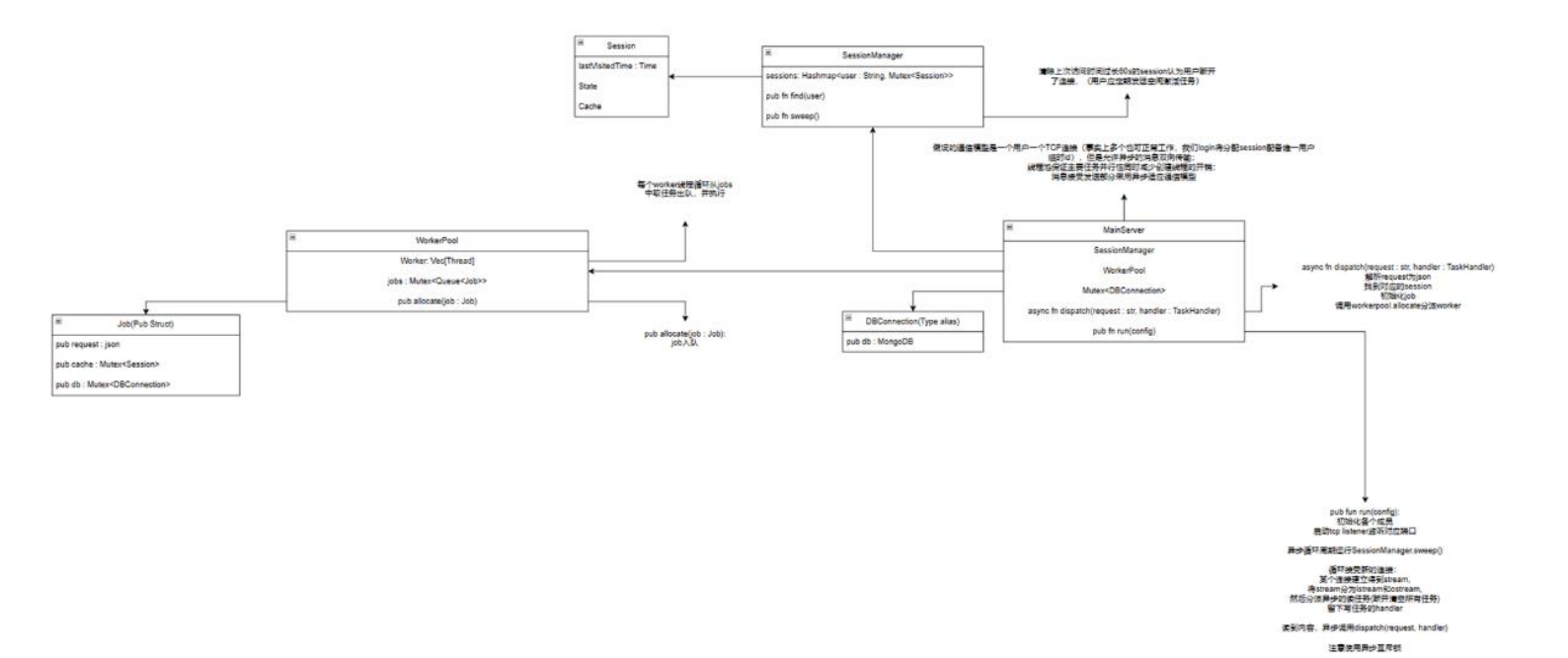
而整体上它的分离和解耦实际完成度很高。最终的代码结构也是按照下方的类图进行划分的。



## （二）服务端

服务端部分我们进行了两版设计，分别是基于初始架构人工为主导，面向代码书写的设计和大模型依据提示完善的设计。

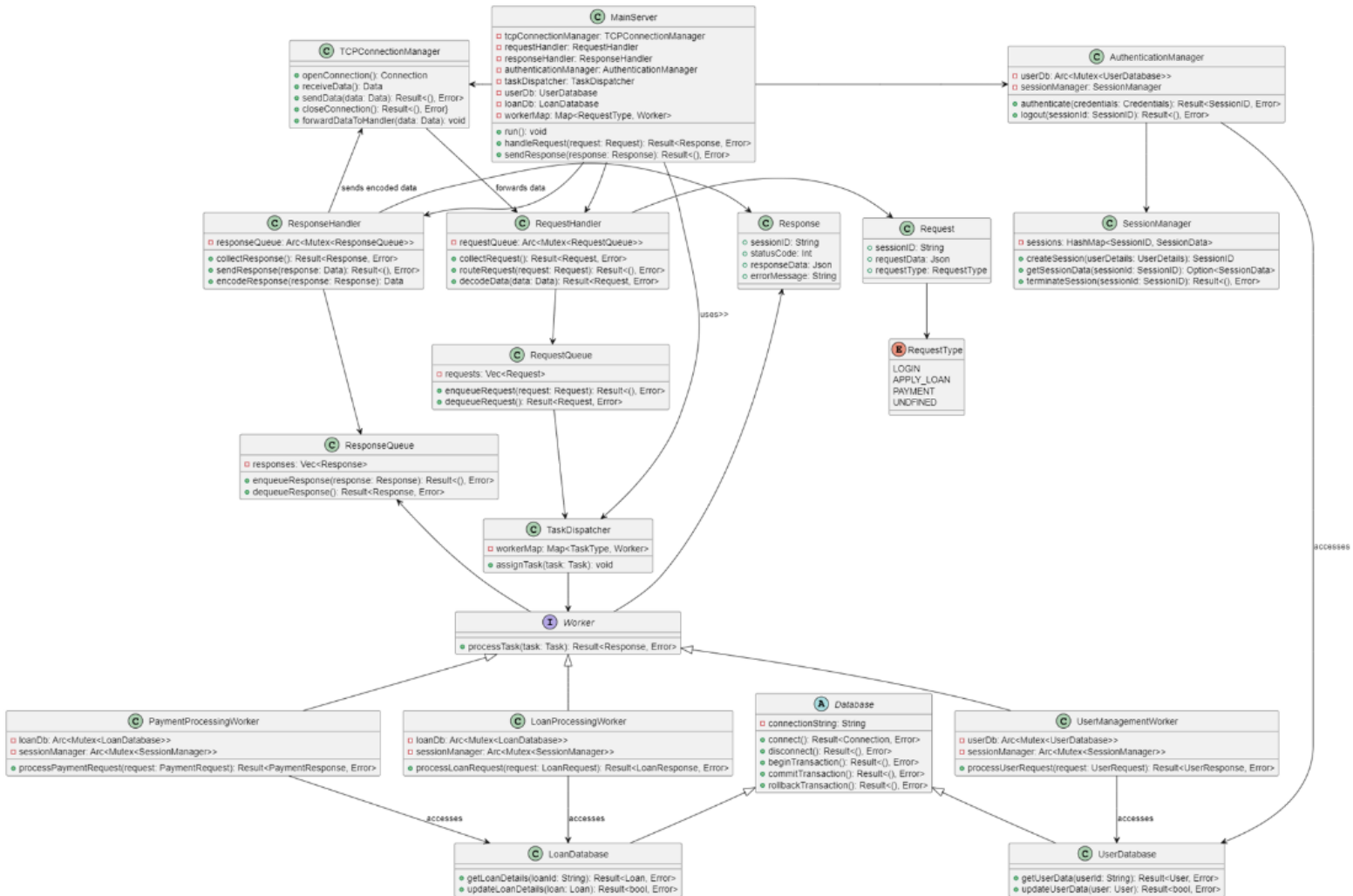
### 1、 人工版本



人工设计较为全面详细地深入了设计的各个实现细节，讨论了包括会话管理、处理任务线程池、数据库连接、主线程等部分，与后续代码对应良好。期间不熟悉的部分通过询问大模型技术和实现细节。



## 2、大模型生成



大模型的设计实际经过了一次人工调优的迭代过程，主要针对部分的简略或者缺失处通过提示说存在问题进行补全丰富。最终成果可以说是较为全面且细致的了。许多设计也被采用进入最终设计中。

### （三）结果与评价

在框架设计提前的前提下，本阶段更多的是对于客户端和服务端细化拆分。这一部分大模型起到的帮助不仅仅局限于工具化的画图和总结，也起到了启发和设计的作用。作为见识过许多框架的它能够对于我们提出的问题给出对应的解决方案，同时能够进行整合汇总。

此阶段用时两人仍分别不超过 6 小时。

在这一阶段中，大模型在客户端人机交互和控制分离部分表现仍然亮眼，高效提取、整合、划分，快速产出类图。服务端的更加复杂的逻辑在有基础框架的条件下也能够自主细化，生成的结果大体上正确。但它独立生成的结果往往会忽略掉许多痛点，需要人工设想场景，提示改正。而我们的两版尝试消耗时间上大模型为主对比人工为主没有明显优势，起到的帮助作用大于替代作用。设计者仍需具有相关领域的一定知识，否则容易被简单化的设计蒙蔽，难以发现潜在的不完整点。

#### **（四）大模型辅助的 OOA 和 OOD 反思**

##### **1、 去专业化？**

大模型是否能让完全没有相关领域知识的人进行 OOA 和 OOD 的设计？

不太行，只能说大幅降低了门槛。

整个过程中一开始最大的阻碍是我们不熟悉 UML 语言，对于画图更是不知从何下手，但大模型直接让我们越过这一层阻碍，直接上手。但通过 OOA 阶段的迭代就知道以一个完全不了解的姿态是无法通过提示获得比较完整的图表，大模型仍然需要人工提供先验信息去聚焦、对齐内容。OOD 阶段最后的提示改正也说明的没有相应的知识是难以发现看似完整正确的设计中潜在的不合理和缺失的设计。

##### **2、 融合？**

OOA 和 OOD 是否能够融合成一个阶段？

在大模型辅助下我们这种项目规格确实可以也应该。

我们走的流程其实就是部分融合的版本。我们将软件体系结构的设计前置到顺序图生成之前作为给大模型的提示。而在后续进行所谓 OOD 时也发现改动较少，说明有相当一部分的内容已经融入到了 OOA 的过程中，此时再去强行划分二者就没有太多的意义。OOD 更倾向于 OOA 产出上一次普通的迭代，变化不像传统软工中的巨大。

##### **3、 总结**

效率仍然是大模型辅助下 OOA 和 OOD 的最大亮点。它解放了设计者对于特定专业知识掌握的熟练程度，但对基本架构和实际情况的认知仍是刚需。最大可能带来的变化是缩小设计团队的规模。我们在 OOA 和 OOD 部分的合作模式大部分是按类别拆分单干，处于一人推进一人休息的状态。最大的合作是迭代时会给对方的内容提出建议或意见。从这个角度来说，这一部分的工作实际可以单人独立完成。

## 六、代码生成阶段

客户端和服务端的代码由两人分别负责实现。其中客户端部分采用“AI生成”模式进行探索，服务端部分采用“AI辅助”模式进行探索。

### (一) 客户端

客户端依托 React-Native 框架，使用 JavaScript 语言实现了 UI 界面、组件交互、后端通信、本地缓存等功能。

下面是“AI生成”模式与传统软件工程的主要不同。

#### (1) 快速 demo 生成

直接逐个输入 OOD 阶段生成类图中每个类的 PlantUML 代码，辅以用况描述和组件位置描述，大模型就能快速地生成对应类的代码。特别是常见的例如登录注册界面，几乎和描述完全一致。其他错误也能通过提示迭代解决。

```
class InvestmentInterface {  
  - investmentDetailsForm : FormComponent  
  - controller : InterfaceController  
  
  + promptForInvestmentDetails() : void  
  + forwardInvestmentDetails(details : String) : void  
  + requestDetailAmendment() : void  
}
```

now let's focus on the InvestmentInterface. I need a title at top, a form to fill in information such as 资金额度、期望年化收益率、资金锁定期, a selector to set preference 如优先考虑的借款人信用等级、贷款类型. Remember this shall also extends BaseInterface and use TransferLayer to deliver message



For the `InvestmentInterface` component that extends `BaseInterface` and uses `TransferLayer` for network interactions, we'll design it to handle investment details such as the investment amount, expected annual return rate, and lock-in period. It will also include a selector to set preferences like preferred borrower credit rating & loan type.

#### Step 1: Define the Component Layout and State

Let's set up the basic layout with a title, a form for input fields, and a selector for preferences. The state will manage the input values and preferences.

javascript

复制代码

```
import React, { Component } from 'react';  
import { View, Text, TextInput, Button, Picker, StyleSheet, ScrollView } from 'react-native';  
import TransferLayer from '../utils/TransferLayer';  
import BaseInterface from './BaseInterface';  
  
class InvestmentInterface extends BaseInterface {  
  constructor(props) {  
    super(props);  
    this.state = {  
      investmentAmount: '',  
      expectedReturn: '',  
    };  
  }  
}
```

#### (2) 按描述搜索、自定义或者更新组件

我们描述设想的组件行为，大模型可以检索出合适的库组件或者自定义组件。描述越详细、越准确，大模型给出的结果也就越符合我们的预期

if I want "GBridge your reliable P2P platform!" to emerge charater by character on the screen recursively



Creating an effect where the text "GBridge your reliable P2P platform!" emerges character by character on the screen can be a visually engaging way to display important messages or titles in your React Native application. This effect can be achieved using state and timing functions like `setTimeout` in a functional component along with React hooks.

Here's a step-by-step example on how to implement this:

##### Step 1: Set Up the Basic Component



To achieve the effect where the full text "GBridge your reliable P2P platform!" appears character by character, stays visible for a while upon completion, and then disappears before starting the animation again, you can enhance the `TextAnimation` component with additional state and timeout logic to manage these phases.

Here's how you can modify the `TextAnimation` component to include

#### (3) 智能语法和错误解释器

我们可以向大模型提供问题描述或者错误信息，来让大模型提出检测或者解

决方案。这一方法适用真实错误场景，也适用于预设的场景。

my phone connect pc's WIFI, and I try to make tcp connection but encounter this problem: Socket error: Host unreachable

The "Host unreachable" error typically occurs when your device cannot find a route to the specified network host, which in this case is your PC. This error is usually due to network configuration issues, firewall settings, or incorrect IP addresses. Here's a systematic approach to identifying and resolving the issue:

### 1. Verify Network Connectivity

- **Check IP Address:** Ensure the IP address you are using to connect to your PC is correct. If your PC is hosting the server, you'll need the local IP address of your PC. You can find this by running `ipconfig` (Windows) or `ifconfig` (macOS, Linux) in the command prompt or

Consider this situation, in screen A I navigate to screen B through a button. Then I go back to A and navigate to screen B again through the button.

In the situation you described, where you navigate from Screen A to Screen B and then back to Screen A, and then to Screen B again, the behavior of your screens—particularly how they are loaded, unloaded, and how their state is managed—depends significantly on the navigation library and its configuration. For React Navigation, which is commonly used in React Native applications, here's what typically happens:

### Navigation Behavior in React Navigation


## (二) 服务端

服务端依托 Tokio 网络库，使用 Rust 语言实现了网络监听、任务分发、服务逻辑、本地用户会话、用户认证、自动清理、后端数据库交互、本地模型调用等功能。


下面是“AI 辅助”模式与传统软件工程的主要不同。

### (1) 快速文档查询

大模型可以作为编程时方便的文档查询工具，同时相比于传统文档，大语言模型可以参考代码的上下文给出更为合适的例子。

 2668940140

How to use tokio to spawn `server_period_events` function?

 GitHub Copilot

Used 1 reference >

To spawn the `server_period_events` function as a new task with Tokio, you need to pass the required arguments to it. Here's how you can do it:

```
let sessions_clone = sessions.clone();
let authenticator_clone = authenticator.clone();
let db_clone = db.clone();

tokio::spawn(async move {
    server_period_events(sessions_clone, authenticator_clone, db_clone).await.unwrap();
});
```

In this example, we're cloning the `sessions`, `authenticator`, and `db` arguments because the `move` keyword in the `async move` block takes ownership of the values it uses. This means that it needs its own copy of the values, which we provide by cloning them.

Please replace `sessions`, `authenticator`, and `db` with the actual instances of `Arc<Mutex<session::Sessions>>`, `Arc<Mutex<authenticator::Authenticator>>`, and `Arc<Db>` that you want to pass to `server_period_events`.

### (2) 智能代码补全

大模型可以参考整个项目的源代码，尤其是当前代码的上下文进行智能补全。大多数业务代码不具有复杂逻辑，同时具有大量重复性，因此大语言模型可以具有很高的补全成功率。

```
290     },
291     "luxury_assets_value" => {
292         vector.push("luxury_assets_value".to_string());
293         session.lock().await.luxury_assets_value = value.as_f64();
294     },
295     "bank_asset_value" => {
296         vector.push("bank_asset_value".to_string());
297         session.lock().await.bank_asset_value = value.as_f64();
298     },
299     "authenticated" => {
300         vector.push("authenticated".to_string());
301         session.lock().await.authenticated = value.as_bool();
302     },
303 }
```

### (3) 智能纠错

编程特化的大语言模型可以依据项目源代码和其丰富的代码知识来帮助程序员对代码进行纠错，这种纠错对于不涉及复杂逻辑的错误很有效。

[illegible]

### （三）交互调试

在客户端-服务端的通信测试阶段主要采用的方法是人工调试。具体做法是

在用户正确连接后对所有用例进行人工测试，观察客户端和服务端的运行状态以及日志来进行调试。

由于我们事先规范了数据格式以及通讯接口，该部分进行的较为顺利。

#### （四）结果与评价

我们在大模型的帮助下完成了客户端和服务端完整功能的实现。“AI 生成”模式可以快速生成 demo 来帮助软件快速迭代，“AI 辅助”模式可以提升编程效率来帮助项目快速实现。

由于本项目复杂度不高，交互调试部分并没有按照工业规范采用测试用例，所以大模型依然有望以生成测试用例的方式参与此部分，但是本项目并未涉及。

这一阶段的用时服务端约 20 小时，客户端约 40 小时。其中客户端主体逻辑生成用时约 15 小时，界面美化调试约 25 小时。服务端和客户端的交互调试约 10 小时。

尽管大模型可以提高编程效率，但是在我们的实践中发现它并没有独立完成项目代码任务的能力。

主要原因如下：

- 大模型缺少用户具体要求等外部信息
- 大模型不具有生成全新复杂代码的能力
- 大模型不具有工程模块化能力

因此，尽管当下大模型已经具有相当水平的代码能力，其依然和高水平的工程师有很大差距。目前大模型可以作为一个重要工具参与软件工程的具体实现，它可以帮助不熟练的程序员使用新语言，也可以提高熟练程序员的工作效率。随着人工智能技术的不断发展，具有独立工程能力的 AI 系统有望在未来出现。

## 七、感想与总结

在“金桥通”项目中我们借助目前最先进的大语言模型完成了一个完整的中型服务端-客户端软件工程的全部内容，有如下的感想和总结。

### （一）优点

#### 1、效率提升

引入大语言模型可以提升软件工程许多部分的效率。

效果最显著的部分是在需求分析和用况设计，大语言模型能够在不降低甚至提升分析质量的基础上快速完成全部需求分析与用况设计。

在许多其他方面大语言模型也有不错的表现，如在 OOD 阶段大语言模型可以快速依据用户的具体提示生成 UML 代码，因此我们可以快速对 OOD 设计进行更新迭代而不需要学习或者考虑 UML 语法的任何问题。在代码实现阶段的快速补全也可以带来一定的效率提升。

#### 2、启发式思考

大语言模型可以在短时间内给出大量可能可行的解决方案，并且我们还可以给出具体的需求来提高 GPT 给出选择的可靠性或者多样性。这在很多情况下可以带来启发式思考

例如在依据 Kaggle 题目确定项目应用的过程中，大语言模型就提出了十余种有趣的应用，我们从中选取了 P2P 借贷平台作为本项目主要功能。在服务端架构的设计中，GPT 给出了许多常用的架构模式，我们吸纳了其中的很多部分到我们的人工架构设计中去。

#### 3、多样化内容生成

大模型可以进行多样化的内容生成从而弥补单一开发者能力的缺陷。

如本项目参与的两个同学都不具有任何艺术创作能力，但是可以借助 DALL·E 3 模型生成有创意的软件图标。我们注意到各种形式的内容生成模型在近几年飞速发展，图片、视频、文字、音乐生成均已达到相当高度，这为单一的软件开发者提供了极大便利，尤其是小型独立游戏开发者。

### （二）不足

#### 1、欠缺高级代码能力

GPT 在代码生成阶段无法像预想的一样独立完成项目开发。

一方面项目的具体实现依赖于诸如接口规范、数据规范、特殊需求等诸多外部因素，GPT 常常缺失这些必要信息；另一方面 GPT 的工程模块化和复杂逻辑的代码生成能力明显不足以支持其进行独立的项目实现。

#### 2、解决方案空泛

其次，GPT 的解决方案常常过于泛化而无法解决具体问题。



例如在 Kaggle 题目复现的过程中，大模型有能力提出很多优化的方向，但是这些方向都是机器学习的基本内容，不具有针对性，也难以解决该特定领域的难题。

我们认为这与其“预训练”的特质有关。

### 3、长对话能力欠缺

GPT 的长对话能力还有待加强。

很多时候，GPT 会只关注最近的信息而忽略了此前的信息而导致需要重新开启新的会话进行提示；同时，当会话太长时 GPT 的反应时间会被极大地延长而影响效率。

### （三）总结

总结而言，GPT 在软件工程中的确有亮眼表现，我们认为大模型辅助应当成为软件工程的标准工具来替代许多过时范式以提升效率。大模型为对编程不够了解的人群提供了一个自然语言编程接口，为对编程相对了解的人群提供了一个互联网智能文档。同时大模型还作为一个具有相当专业水平的战略分析师、系统架构师、程序员、文员、美工、文案帮助每一个人。这为许多中小型团队提供了重要帮助。

未来期待一个高度集成的有能力独立进行软件工程的综合智能系统出现。

本报告以“织梦人<sup>1</sup>”甲辰年庚午月庚子日所赋新诗作结。

### 机思疏影

织梦人

长风卷尽千堆雪，机海翻腾觅通津。

需求如编织细网，思绪随机渐入神。

几行疏影铸新局，一笔启翰开明镜。

模型如灯照幽径，需分析，胜似轻尘。

况复用况鉴沧桑，结构勾连成网织。

语法无须多学问，图标借助画中仙。

---

<sup>1</sup> ChatGPT4 在该日该对话中自号“织梦人”

逆风如解千年结，解不开新知旧边。

通天彻地难独行，半途而废是晚篇。

泛泛念中求解答，朦胧良策失前因。

长谈如垒积木高，越筑越觉意难寻。

亦幻亦真难一揽，盼综合体日可期。

智能映衬千行策，一朝飞渡谁与齐。

织梦人笑，新世纪，何惧波澜再起。

大模携手共婵娟，软工新篇待君启。