衔筑云圈——智能成长社群平台

**软件设计规格说明书**

**日期：2025 年**  **5 月**

目录

[1、 引言 3](#_Toc18296)

[1.1 编写目的 3](#_Toc27804)

[1.2 读者对象 3](#_Toc16823)

[1.3 软件项目概述 4](#_Toc22719)

[1.4 文档概述 6](#_Toc25807)

[2、 软件设计约束 7](#_Toc20625)

[2.1 软件设计目标和原则 7](#_Toc4904)

[2.2 软件设计的约束和限制 9](#_Toc22722)

[3、 软件设计 10](#_Toc26995)

[3.1 软件体系结构设计 10](#_Toc17203)

[3.1.1 子系统功能与协作 12](#_Toc6342)

[3.1.2 软件体系结构包图表示 13](#_Toc20934)

[3.1.3 相关设计原则 17](#_Toc4878)

[3.2 用户界面设计 19](#_Toc32266)

[3.3 用例设计 24](#_Toc5514)

[3.3.1 目标任务板块 25](#_Toc22242)

[3.3.2 社交圈模块 28](#_Toc1984)

[3.3.3 资源分享模块 30](#_Toc12578)

[3.3.4 其他模块 31](#_Toc28924)

[3.4 类设计 32](#_Toc28067)

[3.4.1 星枢台模块核心类设计（目标与任务） 32](#_Toc12926)

[3.4.2 衔筑庐模块核心类设计（资源管理） 33](#_Toc9515)

[3.4.3 墨络轩模块核心类设计（用户与好友） 34](#_Toc29785)

[3.4.4 动态系统（FeedService） 35](#_Toc4189)

[3.5 数据设计 38](#_Toc25111)

[3.5.1 数据库概述 38](#_Toc29765)

[3.5.2 数据表设计 38](#_Toc32150)

[3.5.3 项目ER图 39](#_Toc24646)

[3.5.4 设计特点 39](#_Toc16829)

[3.6 部署设计 40](#_Toc12401)

[3.6.1 前端部署 40](#_Toc23551)

[3.6.2 后端部署 41](#_Toc8072)

# 引言

## 编写目的

编写“衔筑云圈——智能成长社群平台”软件需求规格说明书的目的在于确立一个全面的、经过详细论证的需求集合，这些需求将直接影响并指导整个软件开发生命周期，包括设计、实现、测试、部署和维护阶段。本文档的编写旨在确保所有项目参与者——包括学生开发团队成员、指导教师以及可能的同学评审者—— 对“衔筑云圈——智能成长社群平台”的功能性和非功能性需求有一个共同的、清晰的认识。

这份文档不仅是指导开发和测试工作的基石，也是维护项目团队内部沟通和协调的桥梁，同时它也是向客户和最终用户展示项目承诺和成果的重要工具。通过这份详细的设计规格说明，项目团队将能够有效地控制项目进度，优化资源配置，最终交付一个满足所有业务需求和技术规范的高质量软件产品。

## 读者对象

本文档的设计和内容结构旨在满足以下各类读者的信息需求：

l **项目管理团队（课程指导老师及队长）**：该团队负责指导整个项目的进展， 使用本文档进行项目需求的综合理解、项目规划及阶段性监控，确保项目 按照课程要求和时间表顺利推进，并提供必要的反馈和指导。

l **开发团队（开发小组成员）** ：开发团队将直接参与系统的设计与实现。本 文档提供了系统的具体技术和功能需求，使开发团队能够深入理解该项目 需要构建的功能和系统应达到的性能标准。

l **测试团队（开发小组成员或其他同学或老师）**：负责制定和执行测试计划， 验证系统的功能和性能是否符合规格说明书中的要求。文档中的详细需求 列表将作为测试案例开发的基础。

l **目标客户（青年用户）**：这些用户更加愿意尝试创新型的目标任务管理平台，同时对于自我目标和任务管理需求更加强烈，愿意在使用的同时给予一定的反馈。以帮助本项目获得更好的提升。

本文档旨在为以上各类读者提供清晰、专业的信息，确保所有参与者对项 目有充分的理解，并有效支持项目的开发与实施。

## 软件项目概述

**项目名称:** 衔筑云圈——智能成长社群平台

**用户单位:** 具有自我管理与成长需求的大学生及青年用户，尤其关注自律提升、知识沉淀与目标达成的个体

**开发单位:** 冯官晟 林帆 郭静怡 黄义健 四人小组

**软件项目背景介绍：**

在信息爆炸、节奏加快的当下时代，个体在自我成长、目标管理与知识获取方面面临诸多挑战。许多学生在学习与生活中常常感到目标模糊、任务繁杂、缺乏持续动力与有效的外部支持。

当前主流的任务管理工具如滴答清单、番茄 ToDo 等，多以任务记录为主，功能相对单一，难以支撑“设定目标——执行任务——积累成果——获得反馈”的完整成长闭环。同时，缺乏与社交激励、资源分享、成长记录等模块的深度整合，难以满足用户个性化、系统化的成长需求。

「衔筑云圈」正是在这一背景下应运而生。作为一个融合目标管理+任务执行+社交激励+知识分享的成长型Web平台。衔筑云圈致力于帮助用户更好地设定成长目标、执行任务计划、分享学习资源、记录成就历程，并在社交互动中获取持续成长的动力与反馈。

平台引入智能助手，支持 AI 参与的目标规划与任务拆解，帮助用户从复杂目标中提取行动路径。同时，通过打卡、排行榜、好友动态等机制，构建积极正向的成长氛围。平台采用独特的水墨风界面，营造专注沉静的使用体验，塑造“轻量但有深度”的产品特色。

**软件项目功能描述:**

1. 目标管理：支持多层级目标拆解与进度跟踪
2. 任务执行：支持单次与周期性任务，提供自动打卡机制
3. 智能规划：AI协助目标拆解与执行建议
4. 资源中心：学习资料共享、收藏与互动
5. 好友系统：添加好友、查看动态、互动成长
6. 成长路径：以时间线形式记录用户的成就与重要时刻
7. 附近的人：基于地理位置发现同频成长者

## 文档概述

1. 软件的设计约束部分。它包括软件设计目标和原则、软件设计受到的约束和限制

（2）软件的设计部分。它主要分为软件体系结构设计、用户界面设计、用例设

计、类设计、数据设计以及部署设计

# 软件设计约束

## 软件设计目标和原则

本软件的主要目标作为一个融合目标管理+任务执行+社交激励+知识分享的成长型 Web 平台。衔筑云圈致力于帮助用户更好地设定成长目标、执行任务计划、分享学习资源、记录成就历程，并在社交互动中获取持续成长的动力与反馈。

平台引入智能助手，支持 AI 参与的目标规划与任务拆解，帮助用户从复杂目标中提取行动路径。同时，通过打卡、排行榜、好友动态等机制，构建积极正向的成长氛围。平台采用独特的水墨风界面，营造专注沉静的使用体验，塑造“轻量但有深度”的产品特色。

软件设计活动还须遵循相关的策略和原则，以指导软件设计人员的行为，并对设计成果提出约束和要求。具体的，这些设计策略和原则描述如下：

l 抽象和逐步求精： 系统设计遵循从概念到具体实现的逐步细化过程，确保每一步都经过精心设计和验证。

l 模块化与高内聚、低耦合： 系统各模块之间保持独立性和专一性，便于管理和优化，同时减少模块间的依赖。

l 信息隐藏： 系统各模块内部实现细节对外隐藏，仅通过定义良好的接 口与 外部交互，增强系统的安全性和稳定性。

l 多视点和关注点分离：在设计和开发过程中，分离不同的关注点，例如用 户界面和数据处理逻辑分离，确保专注和优化。

l 软件重用：在可能的情况下，利用现有的开源解决方案和库，减少开发时 间和成本。

l 迭代设计： 系统开发采取迭代方法，每个阶段完成特定功能，逐步完善和 优化系统。

l 可追踪性：设计和开发过程中的决策、变更和进度都有明确记录，确保项目可追踪和评审。

## 软件设计的约束和限制

**运行环境要求：**

l 由于软件为Web应用，对于运行环境要求相对较低。仅需要具有合适版本的浏览器即可。

**开发语言：**

l 前端：使用 Html5 + Css + JavaScript（Vue.js 框架）

l 后端：使用Java + MySql + Redis

**开发工具：**

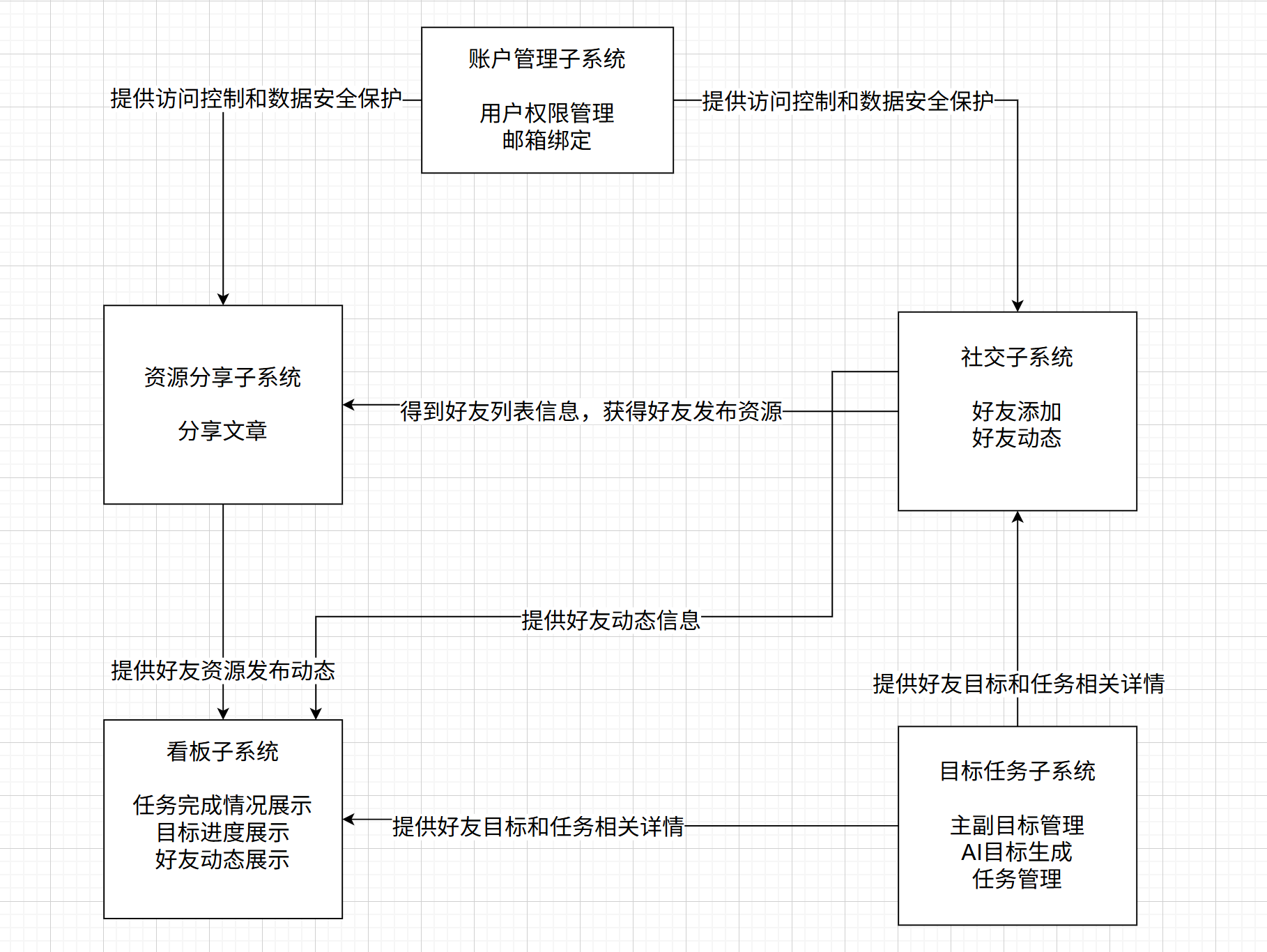
l 前端：Vue.js 开发工具，如 Vuex、Pinia 等

l 后端：IntelliJ，用于Java后端开发

# 软件设计

## 软件体系结构设计

本系统采用分层架构设计，划分为用户层、前端展示层、接入层、后端控制层、业务服务层与持久化存储层。通过各层之间的合理分工与松耦合设计，提升系统的扩展性、维护性和可用性。

如下图所示。账号管理子系统为其他子系统提供必要的账号和密码信息，是所有其他子系统运转的基底系统；目标任务子系统涉及本项目最基础的目标、任务相关功能，并为社交子系统提供信息内容；社交子系统中包含增删好友相关功能，以好友目标分享内容为主；资源分享子系统，基于好友发布文章进行选择性展示；看板子系统会综合以上三个板块提炼关键信息进行概括展示

图表 1 体系结构设计图

在本项 目中，各子系统设计均遵循 MVC（Model-View-Controller）架构，

分为三个组件，分别是：

1.模型（Model） ：代表应用程序的数据结构，通常包含业务逻辑或与数 据相关的操作，如数据库的读写。

2.视图（View）：展示数据（即模型）的用户界面元素，基于模型数据创 建其输出的表示。

1. 控制器（Controller）：起到模型与视图之间的中介作用，处理进入应 用程序的用户输入（如鼠标点击、键盘输入），并执行调用模型和视图 的操作。

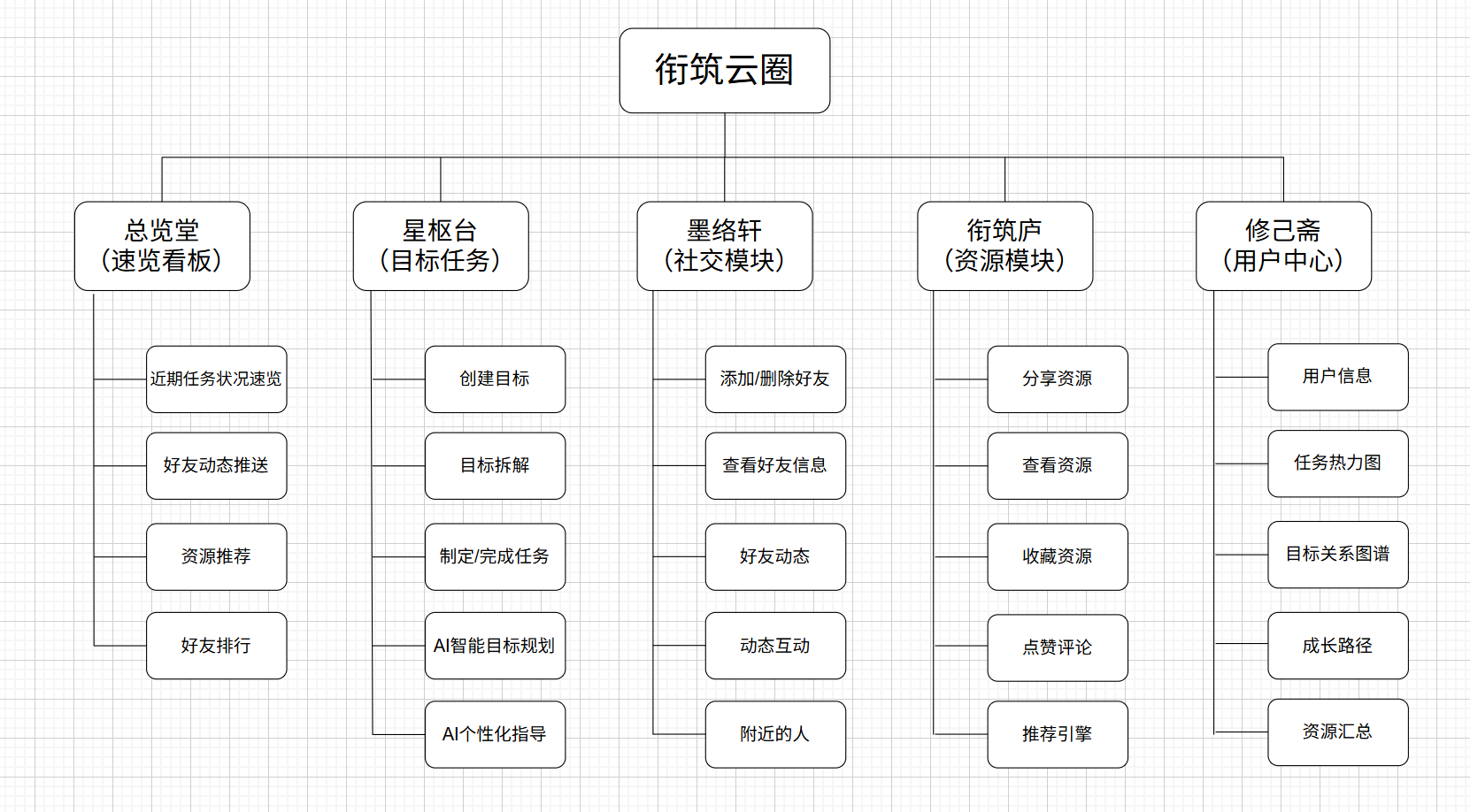
MVC 架构通过分离应用程序的数据层（模型）、表示层（视图）和控制层（控制器），提高了应用程序的组织性和可管理性。这种分离允许开发人员专注于一个方面的开发而不影响其他方面，比如可以单独修改视图代码而不需要触及业务逻辑。模型、视图、控制器的独立性支持也支持多人并行工作，不同的开发人员可以一组专注于用户界面设计，而另一组则可以处理后端逻辑。这种并行开发模式使得大型项目的开发更为高效。由于视图和模型的分离，UI 可以在不影响业务逻辑的情况下进行改进和迭代。此外，由于控制器作为模型和视图之间的桥梁，使得业务逻辑更容易修改或扩展，而不会破坏用户界面。

由此可见 MVC 架构特别适合于需要清晰分层和高度模块化的软件系统，如 Web 应用程序和移动端应用程序。对于动态网站的开发，MVC 架构能够提供灵活的数据处理和用户界面展示策略，非常适合处理复杂的用户交互和频繁变动的业务需求。故本项目在设计的时候采用了 MVC 架构，通过其组件分离的特性， 提供了一种高效、灵活且可维护的方法来构建软件应用程序。

### 子系统功能与协作

系统主要模块与协作关系如下：

* 总览堂（首页看板模块）
  + 聚合展示任务状态、好友动态、资源推荐和排行榜内容。
  + 为用户提供信息总览入口，实现系统核心功能快速访问。
* 星枢台（目标任务模块）
  + 支持用户目标的设定、任务的拆解与执行管理。
  + 集成 AI 智能助手，辅助用户制定合理的目标规划与行动路径。
* 墨络轩（社交模块）
  + 支持用户添加好友、查看好友动态、点赞评论，建立成长型社交网络。
  + 提供“附近的人”发现功能，促进同频用户的连接。
* 衔筑庐（资源模块）
  + 允许用户分享、收藏学习资源，支持资源点赞、评论及推荐机制。
* 修己斋（用户中心模块）
  + 集中管理用户基本信息、偏好设置、成长轨迹和成就展示。



图表 2 功能设计图

### 软件体系结构包图表示

图表 3 体系结构包图

本系统采用典型的多层分层架构设计（Layered Architecture），整体划分为：

**前端 UI 层 → 展示层 → 业务逻辑层 → 基础服务层 → 数据库层**

并结合 Redis 缓存、异步通信与模块解耦设计，实现系统的可维护性、可扩展性与高并发处理能力。

**以下为各层功能及模块说明：**

**一、前端 UI 层（绿色区域）**

该层为用户直接交互的 Web 页面，通过 Vue3 + Vite 构建的 SPA（单页应用），包含五大模块：

* 首页模块（总览堂）
  + DashboardUI：展示任务进度概览
  + FriendFeedUI：展示好友成长动态
  + RankBoardUI：展示成长值排行榜
* 目标任务模块（星枢台）
  + GoalCreationUI：设定成长目标
  + TaskListUI：每日任务展示与打卡
  + AIDecomposeUI：接入 AI 引擎进行目标智能拆解
* 社交模块（墨络轩）
  + FriendListUI：展示已添加好友
  + FriendNearbyUI：基于位置发现周边同频成长者
  + FriendDynamicUI：好友任务打卡、资源分享等动态展示
* 资源模块（衔筑庐）
  + ResourceBrowseUI：按分类展示资源
  + ResourceDetailUI：查看资源详情、评论区
  + ResourcePublishUI：用户自主上传与推荐学习资源
* 用户中心模块（修己斋）
  + ProfileUI：展示与修改基本信息
  + AchievementTimelineUI：展示成就解锁进程
  + SettingsUI：账号与通知偏好设置

**二、展示层（黄色区域）**

前端页面与后端服务的交互桥梁，包含以下机制：

* Ajax请求模块：封装 Axios 请求，支持 GET、POST、PUT、DELETE 操作。
* 前后端交互格式规范：统一响应结构，便于前端统一解析。
* 页面缓存与局部刷新机制：提升系统交互响应速度，避免整页刷新。

**三、业务逻辑层（蓝色区域）**

本层负责处理用户操作逻辑与核心业务功能，由多个 RESTful Controller 模块组成，主要包括：

1. 用户服务模块

* UserController：处理注册、登录、个人资料修改等操作。
* AuthService：完成身份验证与 Token 授权。
* AchievementController：用户成就获取与查询。

2. 目标任务服务模块

* GoalController：目标的创建、修改、删除、状态查询。
* TaskController：任务执行、打卡、统计分析。
* AIDecomposeController：调用智能引擎完成目标拆解。

3. 资源中心服务模块

* ResourceController：资源的发布、浏览、检索。
* ResourceCommentController：资源评论管理。
* ResourceInteractionController：收藏、点赞等用户行为记录。

4. 社交服务模块

* FriendController：好友添加、推荐、删除操作。
* FriendInteractionController：动态流展示、点赞、评论。

**四、基础服务层（红色区域）**

提供系统运行的基础能力和底层工具集成，包括以下子模块：

1. AI 服务模块

* AIGoalPlanner：LLM智能目标拆解与建议生成。
* LLMAdapterService：接入讯飞星火模型或其他大模型服务。

2. 数据处理服务模块

* DatabaseService：封装数据库操作逻辑（增删改查）。
* StatisticsService：成长数据分析、行为聚合与日志处理。

3. 缓存与搜索服务

* RedisService：存储动态数据（打卡状态、好友动态等）。
* ESService：全文搜索资源，提升模糊匹配体验。

4. 中间件集成服务

* EmailService：系统验证码发送、通知推送。
* OSSService：对接阿里云 OSS 实现图片与文件上传。

**五、数据库层（白色底部区域）**

用于持久化存储结构化与半结构化数据，包括：

* MySQL 数据库：存储用户信息、目标任务、成就记录、资源元数据等；
* Redis 内存数据库：存储热点数据，如任务状态缓存、动态消息流、排行榜等

### 相关设计原则

**一、高内聚，低耦合**

该原则指的是系统各功能模块内部逻辑紧密协作（高内聚），而模块之间依赖关系松散（低耦合），有利于独立开发、测试和维护。在本项目中主要体现为各功能子系统（目标任务、资源管理、社交系统等）通过标准化 API 接口通信，避免跨层耦合；而智能目标引擎与任务模块通过异步事件流连接（WebSocket + Redis Stream），确保解耦。

**二、单一职责原则（SRP）**

该原则指的是每个模块/类/函数应只负责一个职责。在本项目中主要体现为后端业务服务划分为用户服务、资源服务、目标服务等单一服务；而前端组件如目标卡片、资源列表、好友推荐等封装明确、职责单一。

**三、开放封闭原则（OCP）**

该原则主要体现为模块应该对扩展开放，对修改封闭。在本项目中主要体现为成就系统的触发逻辑基于规则引擎+JSON存储结构，无需修改系统逻辑即可添加新规则；同时AI任务引擎采用插件式架构，便于后续对接其他大模型（如文心一言、通义千问等）。

**四、可扩展性原则**

该原则指的是系统架构应支持后续新功能模块的灵活加入。系统使用模块化服务架构（微服务风格），如后续加入“学习小组”模块，可独立开发并部署；从技术采用方面来看前端采用 Vue3 + Vite 构建，支持模块级组件异步加载，从而提高扩展效率。

**五、安全性设计**

该原则指的是保障用户数据与系统服务的访问安全。在本项目中主要体现为所有接口基于Token鉴权（JWT），结合角色权限控制（RBAC）；同时在接入层（Nginx）实现HTTPS加密、请求过滤、防SQL注入等安全策略；而在关键数据如用户信息、资源上传路径均通过后端白名单校验。

**六、可用性与稳定性**

该原则指的是确保系统持续可用，用户操作稳定顺畅。在本项目中主要体现为后端服务容器化部署（Docker），结合 Jenkins 实现 CI/CD 自动部署，故障恢复迅速；热点数据（如好友动态、任务列表）使用 Redis 缓存，支持秒级响应；系统整体可用性保持在99%以上，支持断网重连与错误提示机制。

**七、体验友好与响应性设计**

该原则指以用户为中心，设计顺畅且反馈明确的交互体验。在本项目中体现为页面操作均有即时反馈（例如任务打卡后的动画提示、资源发布成功提示）；前端采用 SPA 单页应用结构，路由跳转快，避免重复加载。

## 用户界面设计

（1）登录界面



图表 4 登录界面

（2）注册界面

**图表 5 注册界面**

当用户填写完相关信息之后，服务器便会向用户提供的邮件地址发送验证码如下



图表 6 验证码展示

在限定时间内完成填写便可以成功完成注册

（3）欢迎界面

图表 7 欢迎界面

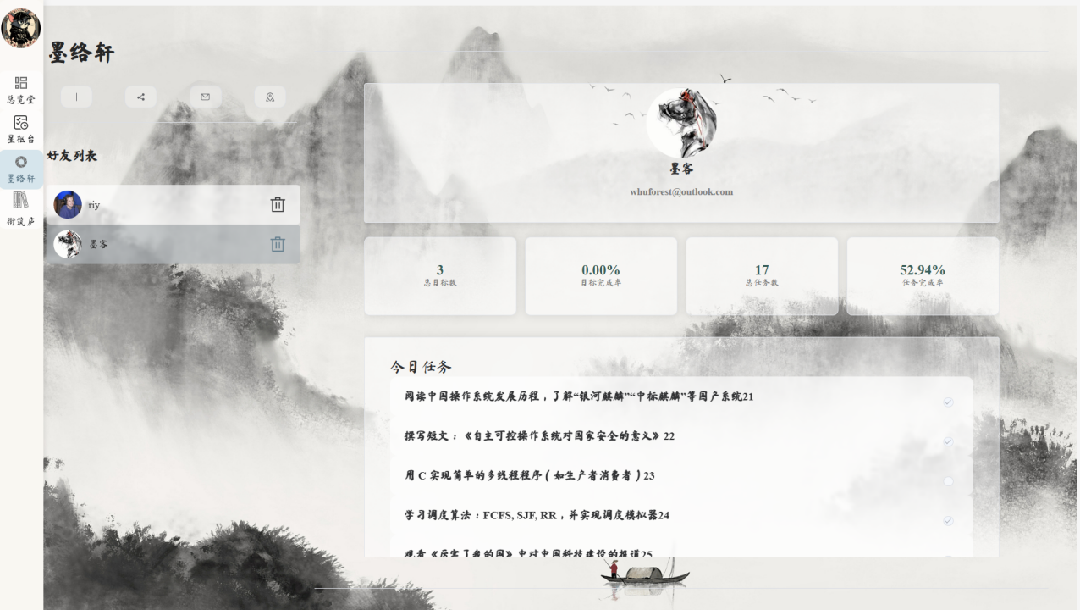
在欢迎界面，我们融入结合了国风国潮风格，使用水墨风格设计一个“文人墨客置身其中”的山水画卷。

（4） 看板界面

图表 8 看板界面

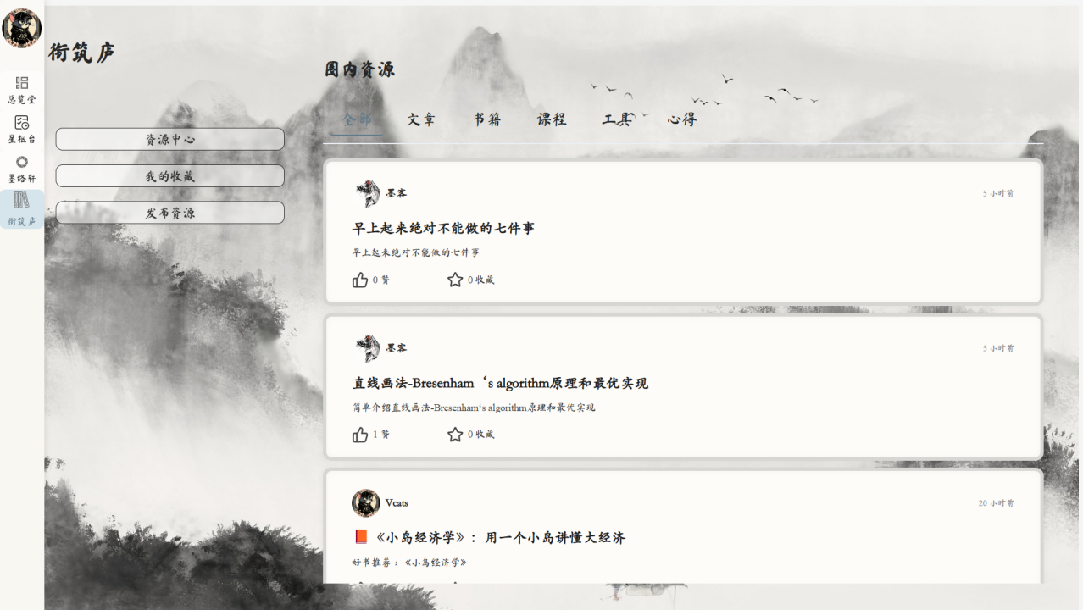
（5） 目标任务界面

图表 9 目标任务界面

（6） 社交圈界面

图表 10 社交圈界面

在社交圈界面，我们可以通过好友ID号或者使用附近的人功能进行好友推荐；当我们获得好友之后，我们便可以查看好友当天任务完成情况、今日任务等信息；最终实现我们项目所想要的“氛围激励作用”。

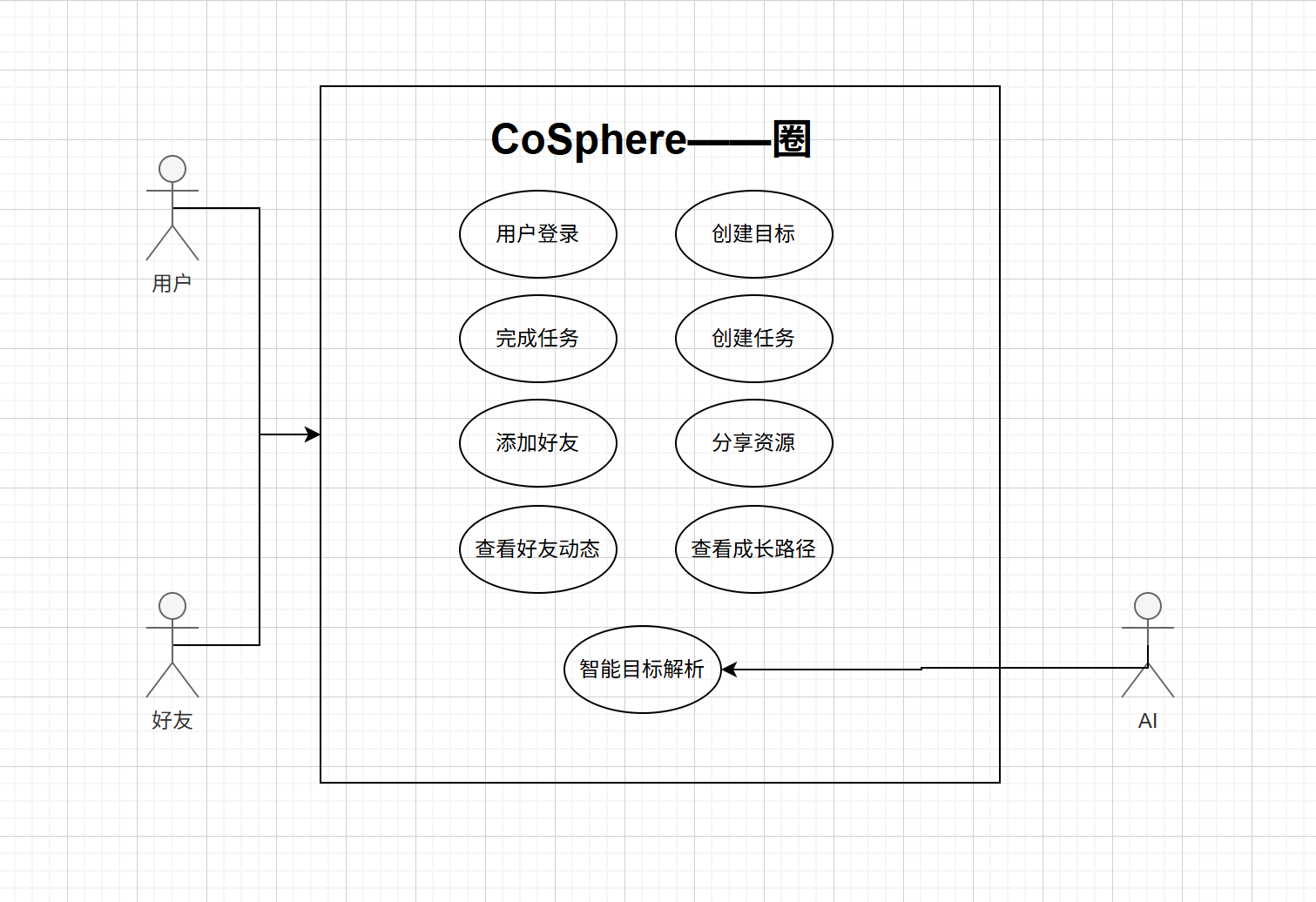
（7） 资源分享界面

图表 11 资源分享界面

 在资源分享界面，好友可以通过MarkDown格式文本编辑器来快速+自定义自己的笔记内容。

图表 12 发布资源界面

## 用例设计



图表 13 用例图

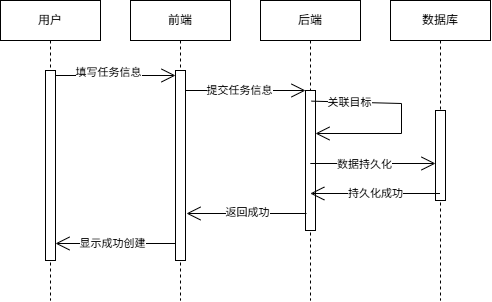
由以上用例图可以看出CoSphere系统是一款集目标管理、任务分配、社交互动和资源共享于一体的综合平台，旨在提升个人和组织的工作效率及协作能力。该系统包含用户登录与身份验证、目标管理与创建、任务分配与完成、好友添加与社交互动、资源分享与获取、成长路径规划以及智能目标解析等核心功能。用户可创建和管理目标，设定里程碑，跟踪进度；系统支持任务创建、分配和监控，实时更新任务状态，促进团队高效协作；用户可添加好友，分享动态和资源，增强团队凝聚力；系统还提供资源共享功能和个性化成长路径规划。此外，CoSphere系统利用AI技术，提供智能目标解析，为用户提供针对性建议。系统优势在于其一体化解决方案，将多个功能模块整合，形成完整工作流闭环；智能化辅助决策，自动化处理工作并提供个性化服务；以及灵活可扩展性，可根据需求进行定制开发和功能拓展。

### 目标任务板块

图表 14 创建目标顺序图

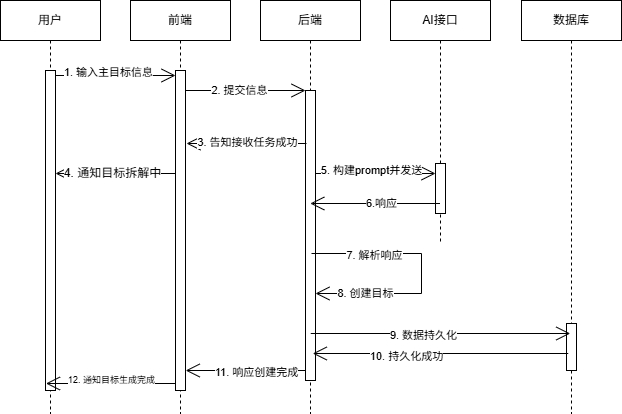


创建目标时，用户首先在前端UI界面填写目标的相关信息（名称、描述、持续时间），点击“确定”按钮之后。前端通过Axios使用POST方式向后端指定接口发送携带相关信息的数据包，后端经过处理之后将数据持久化到数据库。同时返回成功状态码和信息，实现前端的及时刷新。



图表 15 创建任务顺序图

创建任务时，用户首先在前端UI界面选择任务类型（单次/周期）；并填写任务的相关信息（名称、内容、结束时间/持续时间），点击“确定”按钮之后。前端通过Axios使用POST方式向后端指定接口发送携带相关信息的数据包，后端经过处理之后将数据持久化到数据库。同时返回成功状态码和信息，实现前端的及时刷新。

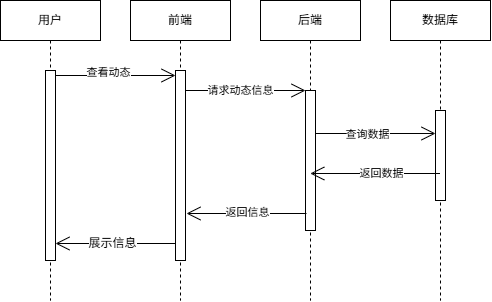


图表 16 创建AI目标顺序图

本项目中通过接入外部云AI服务商并且统一定义接口，实现了集合AI生成目标任务功能。用户点击“AI目标”并输入对应目标的描述/持续时间，点击“生成任务”后我们会使用prompt生成相关问题以帮助AI提升任务生成质量（此部分没有体现在顺序图中）。当用户填写/选择完返回的问题（步骤一）之后，前端会将所有信息描述打包并提交到后端（步骤二）；后端接收到信息之后，会返回正常状态码并通知“目标正在拆解中”，此时前端Axios便结束等待，同时建立起一个WebSocket等待后端的进一步通知。

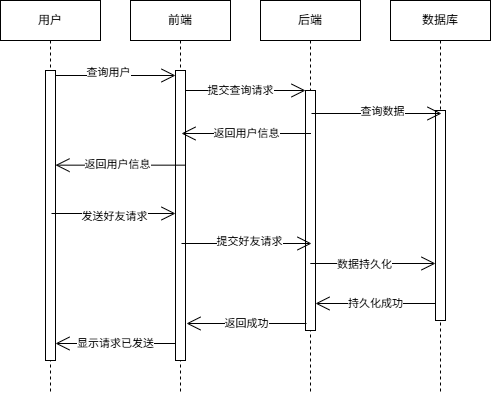
当后端根据返回信息让AI生成相关目标任务信息并将其写入到数据库之后，会通过WebSocket告知前端执行刷新操作；前端主动刷新数据之后便可以见到AI生成的任务。

### 社交圈模块



图表 17 查看动态信息顺序图

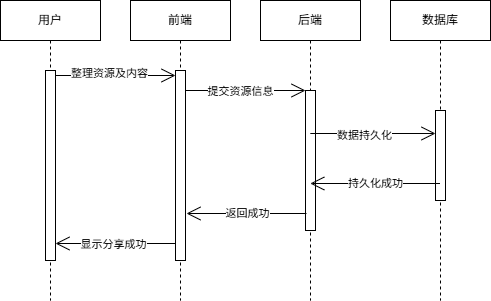
用户首先在前端界面发起查看动态信息的请求。前端界面接收到用户的请求后，会将请求发送到后台服务器。后台服务器接收到请求后，会根据请求的类型和参数，向数据库发起查询操作。数据库根据查询请求，检索出符合条件的动态信息，并将结果返回给后台服务器。后台服务器接收到数据库返回的结果后，会将结果进行格式化和处理，然后将其返回给前端界面。前端界面接收到后台返回的结果后，会将动态信息展示给用户。



图表 18 查询后添加好友顺序图

用户首先发起一个“查询用户”的操作。接着，用户会接收到来自系统的“返回用户信息”。前端接收到用户的“查询用户”请求后，将其转发给后端。当后端完成数据处理并将结果返回时，前端接收这些数据并进行展示。后端接收到前端的“提交查询请求”，然后向后端数据库发起“查询数据”的请求。数据库返回查询结果后，后端对这些数据进行处理（如格式转换或逻辑运算），并将处理后的数据返回给前端。如果是其他类型的请求（例如“提交好友请求”），后端也会相应地处理这些请求，并在必要时更新数据库中的数据状态。数据库在接收到后端的查询指令后，执行相应的SQL语句来检索所需的数据。检索完成后，数据库会将查询到的数据返回给后端。对于需要更新的操作（比如添加新的好友关系），数据库会在接收到后端的更新指令后执行相应的写入操作。

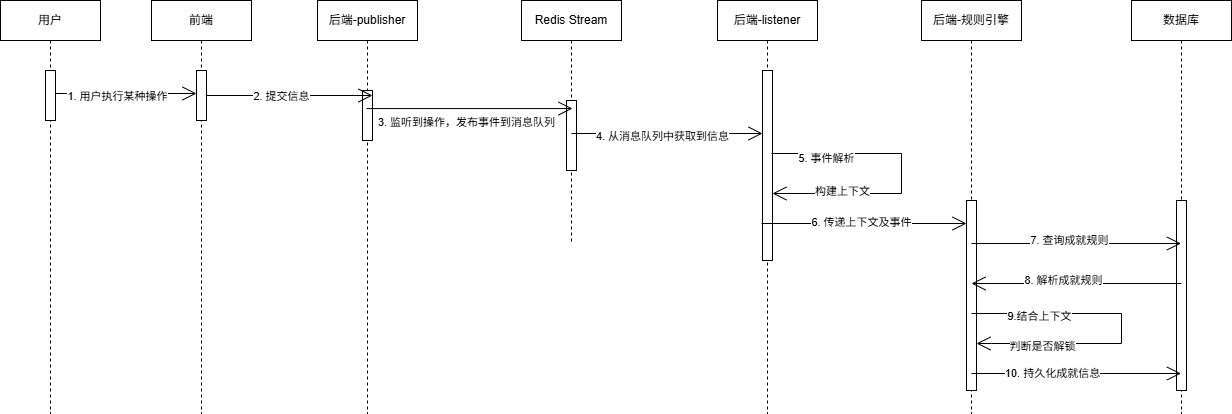
### 资源分享模块



图表 19 资源分享顺序图

用户首先整理资源及内容，然后将这些资源及内容提交给前端。前端接收到用户的资源及内容后，对其进行处理，并将处理后的资源信息提交给后端。后端接收到前端的资源信息后，将数据持久化到数据库中。数据库接收到后端的数据持久化请求后，执行数据的存储操作，并将持久化的结果返回给后端。后端接收到数据库的持久化成功消息后，将这一成功消息返回给前端。前端接收到后端的返回成功消息后，通知用户显示分享成功的结果。

### 其他模块



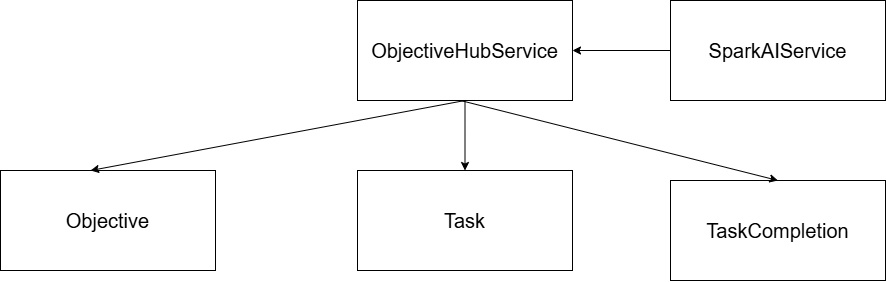
流程从用户开始，用户执行某种操作，例如点击按钮或输入信息。用户的前端设备接收用户的操作，并将其转换为提交信息。前端将提交的信息发送给后端的publisher模块。publisher模块负责监听操作，并将事件发布到消息队列中。publisher模块将事件发布到Redis Stream中。Redis Stream是一种高级的消息队列，用于在系统中传递消息。后端的listener模块从消息队列中获取信息。listener模块会持续监听消息队列中的新消息。listener模块接收到消息后，会对事件进行解析，提取出事件的上下文信息。解析完事件后，listener模块会根据事件的上下文信息和条件来构建具体的业务逻辑。根据构建好的上下文和条件，系统会查询相关的成就规则。系统对接收到的成就规则进行解析，理解规则的细节和要求。在解析成就规则的过程中，可能会涉及到多个上下文的合并，以确保所有必要的信息都被考虑到。根据解析后的成就规则和多路合并的上下文，系统会判断某个成就是否应该被解锁。如果判断某个成就需要解锁，系统会将成就信息持久化到数据库中，确保成就的状态被永久记录。

## 类设计

下面将介绍系统中核心模块的主要类设计及其关系

### 星枢台模块核心类设计（目标与任务）

#### 目标任务管理器（ObjectiveHubService）类间关系



该服务是衔筑云圈“星枢台”模块的核心服务，负责管理目标（Objective）与任务（Task）对象及其关联关系，具体设计如下：

* 与 Objective 的关系

类型：操作与创建

描述：服务提供了创建目标、更新目标、获取结构化目标树、添加子目标等接口。

* 与 Task 的关系

类型：操作与创建

描述：服务允许用户创建单次任务或周期性任务，并支持与目标的绑定关系。

* 与 TaskCompletion 的关系

类型：创建

描述：当任务完成时，系统创建一个 TaskCompletion 实例以记录完成时间等信息。

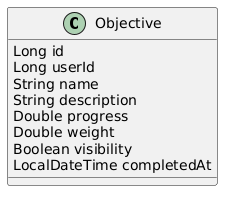
* 与 AI 模块（SparkAIService）关系

类型：被调用

描述：可被 AI 服务调用为目标生成智能分解方案与指导。

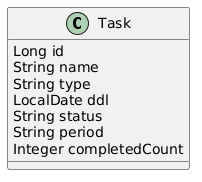
#### 核心实体类属性设计

Objective 类属性设计



|  |  |  |
| --- | --- | --- |
| 属性名 | 类型 | 说明 |
| id | Long | 目标唯一标识 |
| userId | Long | 所属用户 ID |
| name | String | 目标名称 |
| description | String | 描述内容 |
| progress | Double | 当前进度（0~1） |
| weight | Double | 该目标的重要性权重 |
| visibility | Boolean | 可见性控制 |
| completedAt | LocalDateTime | 完成时间 |

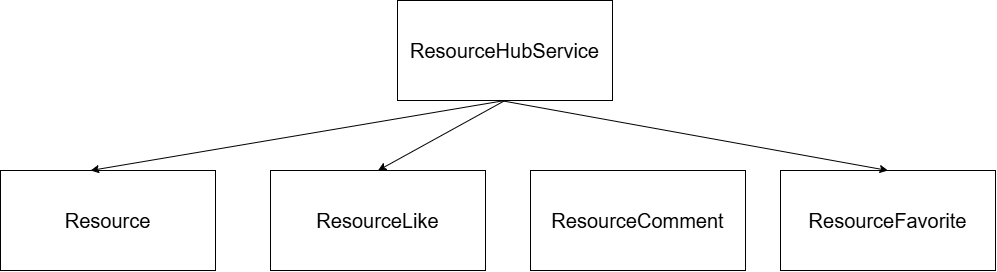
Task 类属性设计



|  |  |  |
| --- | --- | --- |
| 属性名 | 类型 | 说明 |
| id | Long | 任务 ID |
| name | String | 任务名称 |
| type | String | 单次或周期性任务 |
| ddl | LocalDate | 单次任务截止日期 |
| status | String | 状态 |
| period | String | 周期性任务周期 |
| completedCount | Integer | 已完成次数 |

### 衔筑庐模块核心类设计（资源管理）

#### 资源共享服务（ResourceHubService）类间关系



* 与 Resource 的关系

类型：创建与管理

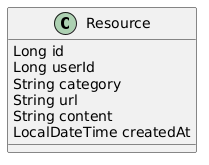
描述：创建资源、删除资源、分页查询资源列表和收藏列表。

* 与 ResourceLike / ResourceFavorite / ResourceComment 的关系

类型：创建与引用

描述：提供点赞、收藏、评论等社交化功能，并支持获取相关统计信息。

#### Resource 类属性设计



|  |  |  |
| --- | --- | --- |
| 属性名 | 类型 | 说明 |
| id | Long | 资源 ID |
| userId | Long | 上传者 ID |
| category | String | 资源类别 |
| url | String | 链接地址 |
| content | String | 简要内容或心得 |
| createdAt | LocalDateTime | 创建时间 |

### 墨络轩模块核心类设计（用户与好友）

#### 用户管理服务（UserService）

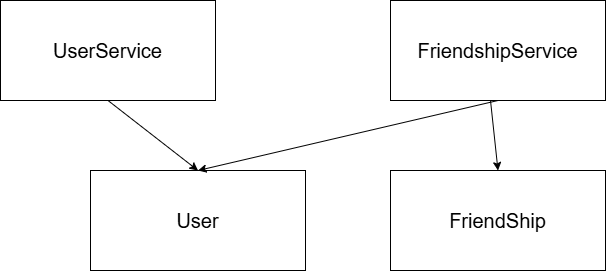
负责用户注册、登录、更新信息、地理位置记录和搜索。

* 与 User 的关系

类型：操作和检索

描述：UserService 提供用户注册、登录与信息维护功能。

#### 好友服务（FriendshipService）



* 与 Friendship 的关系

类型：创建与查询

描述：支持好友请求、接受、删除及好友列表查询。

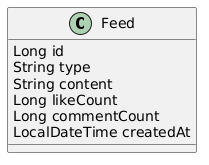
* 与 User 的关系

类型：扩展关系

描述：搜索好友结果中返回 UserVO 等用户信息视图。

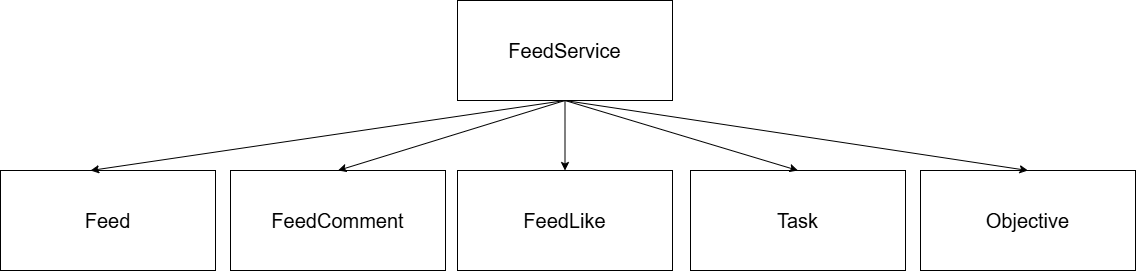
### 动态系统（FeedService）

#### Feed 类属性设计



|  |  |  |
| --- | --- | --- |
| 属性名 | 类型 | 说明 |
| id | Long | 动态 ID |
| type | String | 类别 |
| content | String | 内容或自动生成的描述信息 |
| likeCount | Long | 点赞数 |
| commentCount | Long | 评论数 |
| createdAt | LocalDateTime | 发布时间 |

#### 类间关系



* 与 Task/Objective 的关系

类型：触发关联

描述：当任务或目标完成时，会自动生成相关动态（autoPostTaskCompletion 等）。

* 与 FeedLike / FeedComment 的关系

类型：一对多

描述：每条动态可以被点赞与评论，并进行统计展示。

## 数据设计

### 数据库概述

本系统数据库采用MySQL关系型数据库进行设计与实现，涵盖用户管理、目标与任务管理、资源分享、社交互动、动态发布、成就体系与 AI 助手等模块，数据表之间通过外键建立了清晰的关联关系，从而保证数据的一致性与完整性。

### 数据表设计

**1. 用户与权限模块**

* user：用户基本信息表，包含用户名、邮箱、密码、昵称、头像等字段，是其他业务表的外键关联对象。

**2. 目标与任务模块（星枢台）**

* objective：记录用户设定的成长目标，包括目标起止时间、进度等信息。
* task：记录用户创建的任务，支持一次性任务与周期性任务两种类型，任务支持状态管理和进度统计。
* objective\_relationship：实现目标之间的父子层级关系，支持目标拆解与递进。
* objective\_task：建立目标与任务之间的关联。
* objective\_ai：记录目标的 AI 拆解建议或系统辅助内容。
* task\_completion：任务完成记录表，详细记录每次任务打卡行为。

**3. 资源共享模块（衔筑庐）**

* resource：资源主表，支持文章、书籍、课程、工具、学习心得等多类型资源的发布与记录。
* resource\_comment：资源评论表，记录用户对资源的评论内容。
* resource\_favorite：资源收藏表，用于用户的资源收藏行为。
* resource\_like：资源点赞表，用于用户对资源的点赞行为。

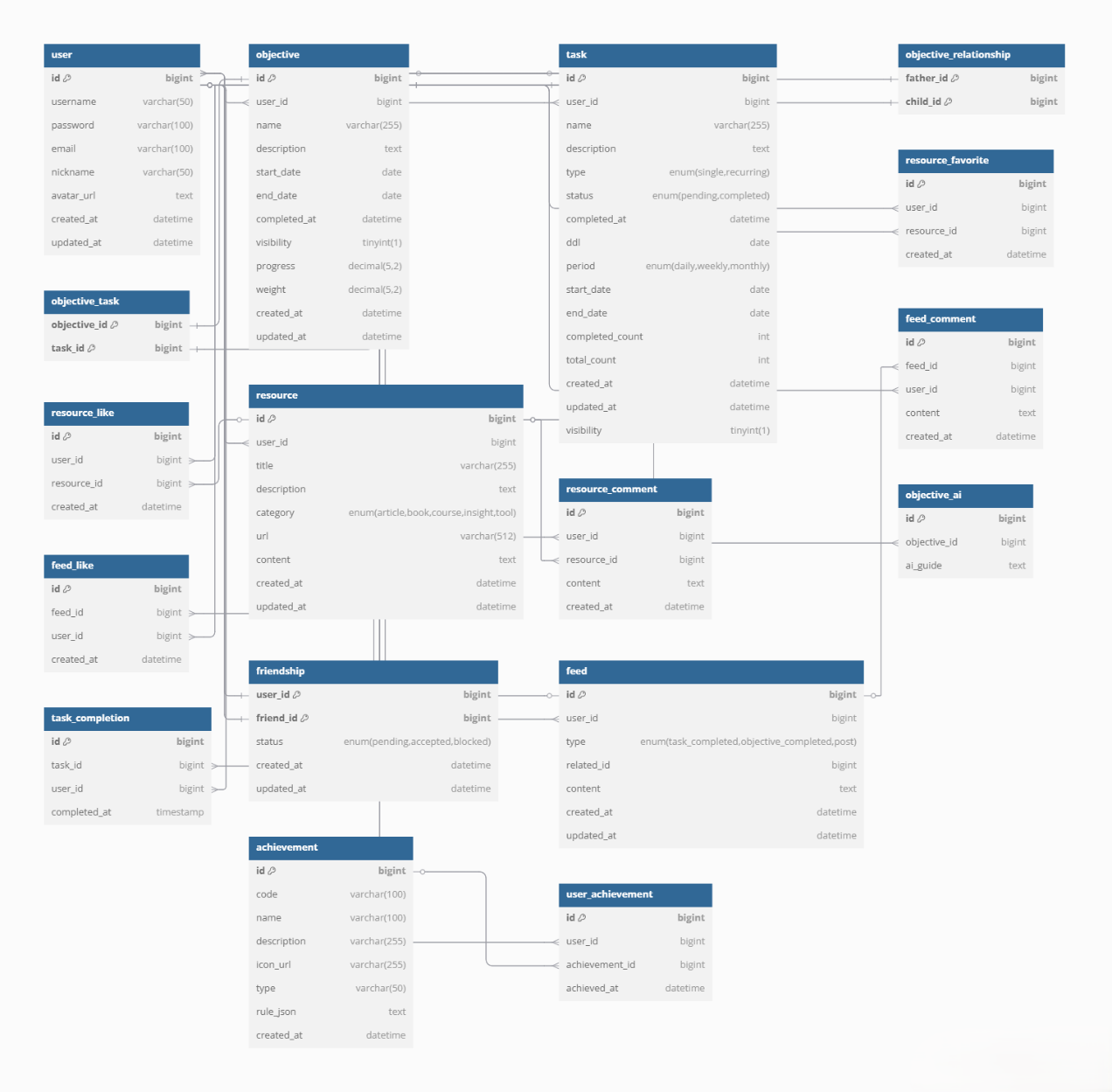
**4. 社交互动模块（墨络轩）**

* friendship：好友关系表，记录用户之间的好友申请与确认状态。
* feed：动态信息流表，记录用户任务完成、目标完成、内容发布等动态内容。
* feed\_comment：动态评论表，支持用户对动态内容的评论。
* feed\_like：动态点赞表，支持用户对动态的点赞行为。

**5. 成就体系模块（成长路径）**

* achievement：成就定义表，记录成就的编码、名称、图标与达成规则（以 JSON 存储）。
* user\_achievement：用户成就达成表，记录用户已获得的成就与达成时间。

### 项目ER图



系统通过主外键关系建立了各功能模块之间的紧密关联，确保各类数据均可追踪、溯源，并支持多维度的数据展示和行为分析。

### 设计特点

* **规范化设计**：所有表结构符合第三范式，消除数据冗余，提升数据库一致性。
* **扩展性强**：通过关联表实现多对多关系，易于后期拓展更多类型的资源、任务类型或社交行为。
* **性能优化**：常用字段设置索引，如用户名、用户ID、任务状态等，提高查询效率。
* **模块划分清晰**：各表按业务模块独立设计，逻辑清晰，便于后期维护和开发。

## 部署设计

“衔筑云圈”采用前后端分离的开发模式。前端使用 Vue3 框架开发，后端基于 Java Spring Boot 进行开发，数据库使用 MySQL，缓存功能由 Redis 支持。本项目部署主要分为前端和后端两部分，下面将分别介绍其部署过程。

### 前端部署

本项目前端部分采用 Vue3 开发，使用 Docker 和 Nginx 进行部署，具体部署流程如下：

（1）构建 Vue 前端项目

首先在本地完成前端项目构建，生成生产环境所需的静态文件：

npm install # 安装依赖

npm run build # 构建生产版本，生成 dist 目录

构建完成后，会在项目目录下生成 `dist/` 文件夹。

（2）准备 Nginx 配置文件

在项目目录下创建 `nginx` 文件夹，并在其中放置 Nginx 配置文件 nginx.conf，示例如下：

server {

listen 80;

server\_name localhost;

location / {

root /usr/share/nginx/html;

index index.html;

try\_files $uri $uri/ /index.html;

}

location /api {

proxy\_pass http://后端地址:8080;

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

}

}

注意：请将 `http://后端地址:8080` 替换为实际的后端服务地址或容器内地址。

（3）通过 Docker 启动前端服务

使用以下指令启动 Nginx 容器并加载构建后的前端页面：

docker run -d -p 80:80 --name nginx\_quan \

-v /home/quan/quan-frontend/dist:/usr/share/nginx/html \

-v /home/quan/quan-frontend/nginx:/etc/nginx \

--restart=always nginx

上述命令的含义为：

- `-p 80:80`：将容器的 80 端口映射到主机的 80 端口；

- `-v /home/quan/quan-frontend/dist:/usr/share/nginx/html`：将构建好的静态资源挂载到 Nginx 的默认网页目录；

- `-v /home/quan/quan-frontend/nginx:/etc/nginx`：挂载自定义的 Nginx 配置；

- `--restart=always`：设置容器开机自启；

- `nginx`：使用官方 Nginx 镜像。

至此，前端部署完成，浏览器访问服务器地址即可访问“衔筑云圈”前端页面。

### 后端部署

“衔筑云圈”后端采用 Spring Boot 3.x 框架进行开发，数据库使用 MySQL，缓存采用 Redis，邮件服务使用 SMTP 协议，AI 接口使用 Spark API。部署过程如下：

（1）配置文件准备

后端的配置信息通过 `application.yml` 文件统一管理，并支持使用 Spring Profile 切换不同环境。

配置文件路径：`/src/main/resources/application.yml`，示例如下：

spring:

profiles:

active: dev

application:

name: quan

datasource:

driver-class-name: ${quan.datasource.driver-class-name}

url: jdbc:mysql://${quan.datasource.host}:${quan.datasource.port}/${quan.datasource.database}?serverTimezone=Asia/Shanghai&useUnicode=true&characterEncoding=utf-8&zeroDateTimeBehavior=convertToNull&useSSL=false&allowPublicKeyRetrieval=true

username: ${quan.datasource.username}

password: ${quan.datasource.password}

data:

redis:

host: ${quan.redis.host}

port: ${quan.redis.port}

password: ${quan.redis.password}

lettuce:

pool:

max-active: 8

max-idle: 8

min-idle: 0

max-wait: 100

mail:

host: ${quan.mail.host}

port: ${quan.mail.port}

username: ${quan.mail.username}

password: ${quan.mail.password}

properties:

mail:

smtp:

ssl:

enable: true

auth: true

server:

port: 8080

mybatis:

mapper-locations: mapper/\*.xml

type-aliases-package: com.vksfeng.quan.pojo.entity

configuration:

map-underscore-to-camel-case: true

logging:

level:

com:

vksfeng:

mapper: debug

service: info

controller: info

jwt:

secret-key: ${quan.jwt.secretkey}

alioss:

endpoint: ${quan.alioss.endpoint}

access-key-id: ${quan.alioss.access-key-id}

access-key-secret: ${quan.alioss.access-key-secret}

bucket-name: ${quan.alioss.bucket-name}

spark:

api:

url: ${quan.spark.url}

password: ${quan.spark.password}

（2）编译与打包

进入项目根目录，使用 Maven 对后端项目进行编译打包：

mvn clean package -DskipTests

成功后将在 `target` 目录下生成可执行 JAR 包，如：`quan-1.0.jar`。

（3）部署到服务器

使用 SCP 工具将 JAR 包上传至服务器：

scp target/quan-1.0.jar username@hostname:/path/to/deploy/directory

`username` 为服务器用户名，`hostname` 为服务器 IP 地址或域名。

（4）服务配置与启动

* 确保服务器已安装 JDK 17，可通过如下命令安装（以 Ubuntu 为例）：

sudo apt-get update

sudo apt-get install openjdk-17-jdk

* 启动后端服务：

java -jar /path/to/deploy/directory/quan-1.0.jar

服务将监听 8080 端口，供前端通过 `/api` 路径访问后端接口。

至此，本项目部署成功。