知你社交——系统设计文档

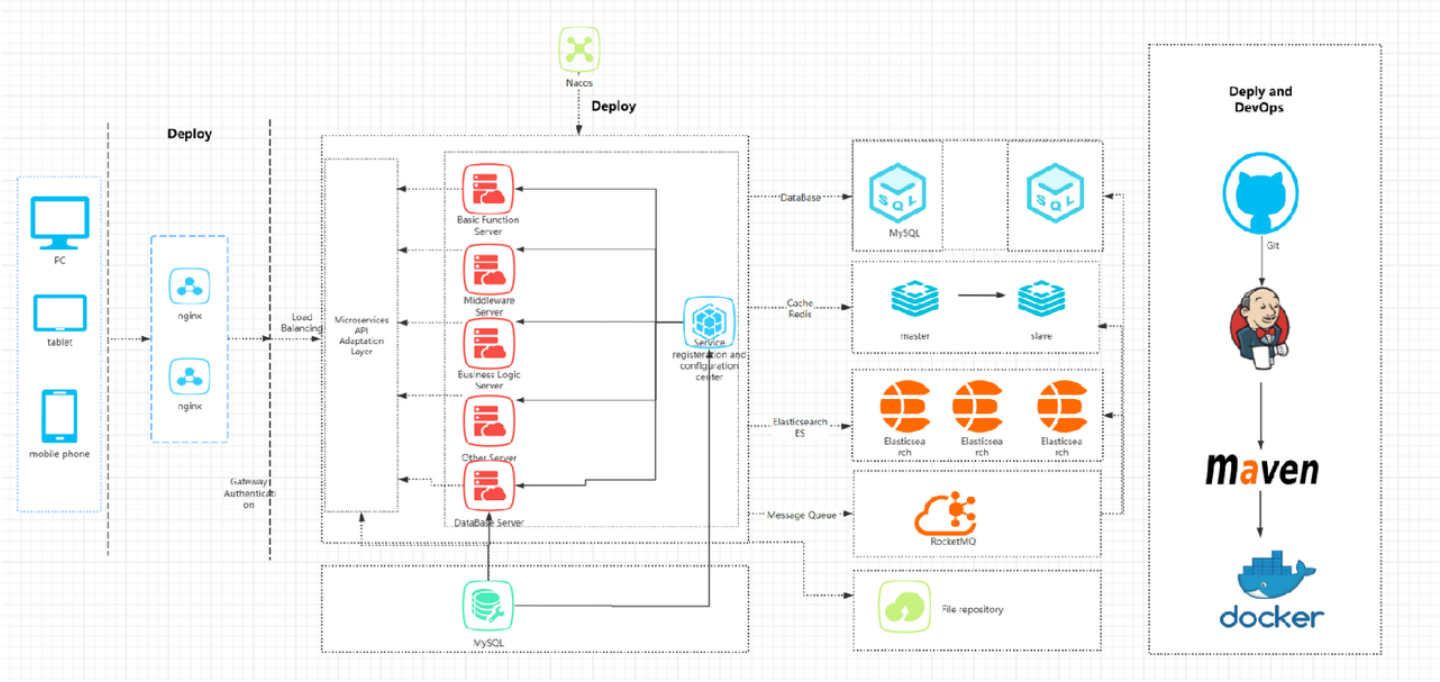
项目成员

组长：2253551 李沅衡

学号	姓名
2253551	李沅衡
2251225	王铭乾
2250763	李俊旻
2254272	赵子毅
2251760	黄志栋

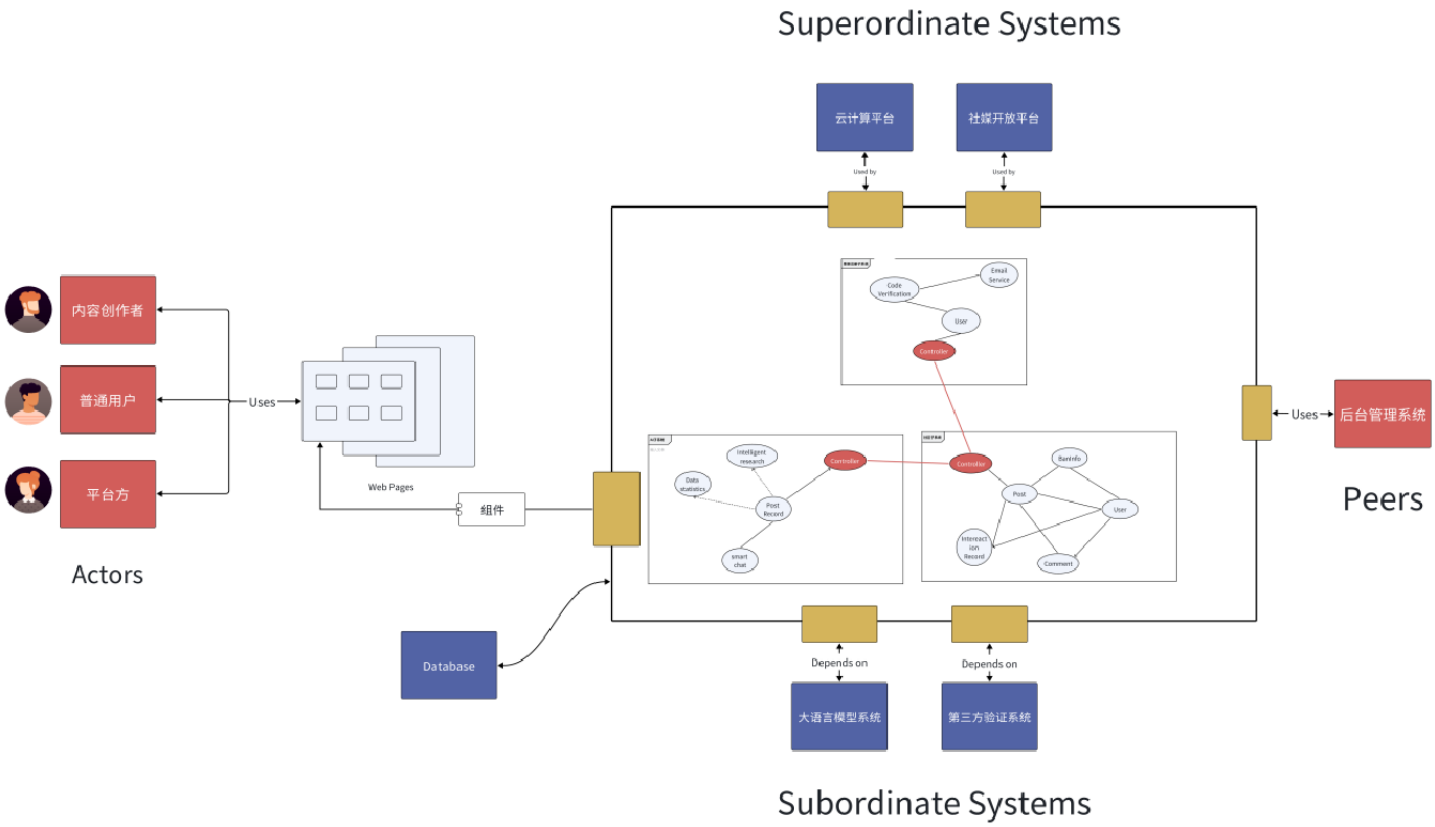
1. 架构设计

1.1 部署设计



系统采用微服务架构并通过Nacos进行服务注册与配置管理，利用Docker和DevOps工具链实现自动化部署和运维。客户端通过PC、Tablet、Mobile Phone等设备经由Nginx负载均衡器及Gateway Authentication访问系统，确保请求的合法性和高效分发。核心服务包括提供基础功能的Basic Function Server、处理通用逻辑的Middleware Server、执行业务逻辑的Business Logic Server以及其他特定功能的服务，所有这些都通过Microservices API Adaptation Layer进行交互，并依赖于MySQL和Redis分别进行数据持久化和缓存加速。此外，系统集成了Elasticsearch用于全文搜索和数据分析，RocketMQ作为消息队列支持异步通信，同时使用File Repository进行文件存储。整个系统的构建和部署流程中应用了Git进行版本控制、Maven进行项目构建，以及通过自动化部署和DevOps实践保证系统的快速迭代和稳定运行。

1.2 ACD设计



1.2.1 参与者

系统中的参与者主要包括内容创作者、普通用户和平台方。内容创作者通过平台发布原创内容，丰富平台资源；普通用户则浏览这些内容，并参与互动，如评论或分享；而平台方负责平台的日常运营与管理，确保所有操作符合规范并维持系统的正常运行。

1.2.2 组件

Web Pages，为用户提供了一个直观的界面来访问系统功能。Controller作为控制层，接收用户的请求并将它们转发给相应的服务进行处理。智能检索模块提高了信息查找的效率和准确性，让用户能够迅速找到感兴趣的内容。数据监控确保系统数据的准确性和完整性。Post Record、Smart Chef、Internal ISP Record、User Info、Post以及Comment等组件分别负责记录帖子信息、提供智能创作建议、管理ISP记录、维护用户资料、管理和展示帖子以及处理评论等任务。

1.2.3 领域模型

Code Verification模块主要用于验证用户身份，保障账户安全。Email Service则提供了邮件发送的功能，支持系统内的通知发送，如注册确认、密码找回等重要信息的传递，保证了用户与系统之间有效沟通的渠道畅通无阻。

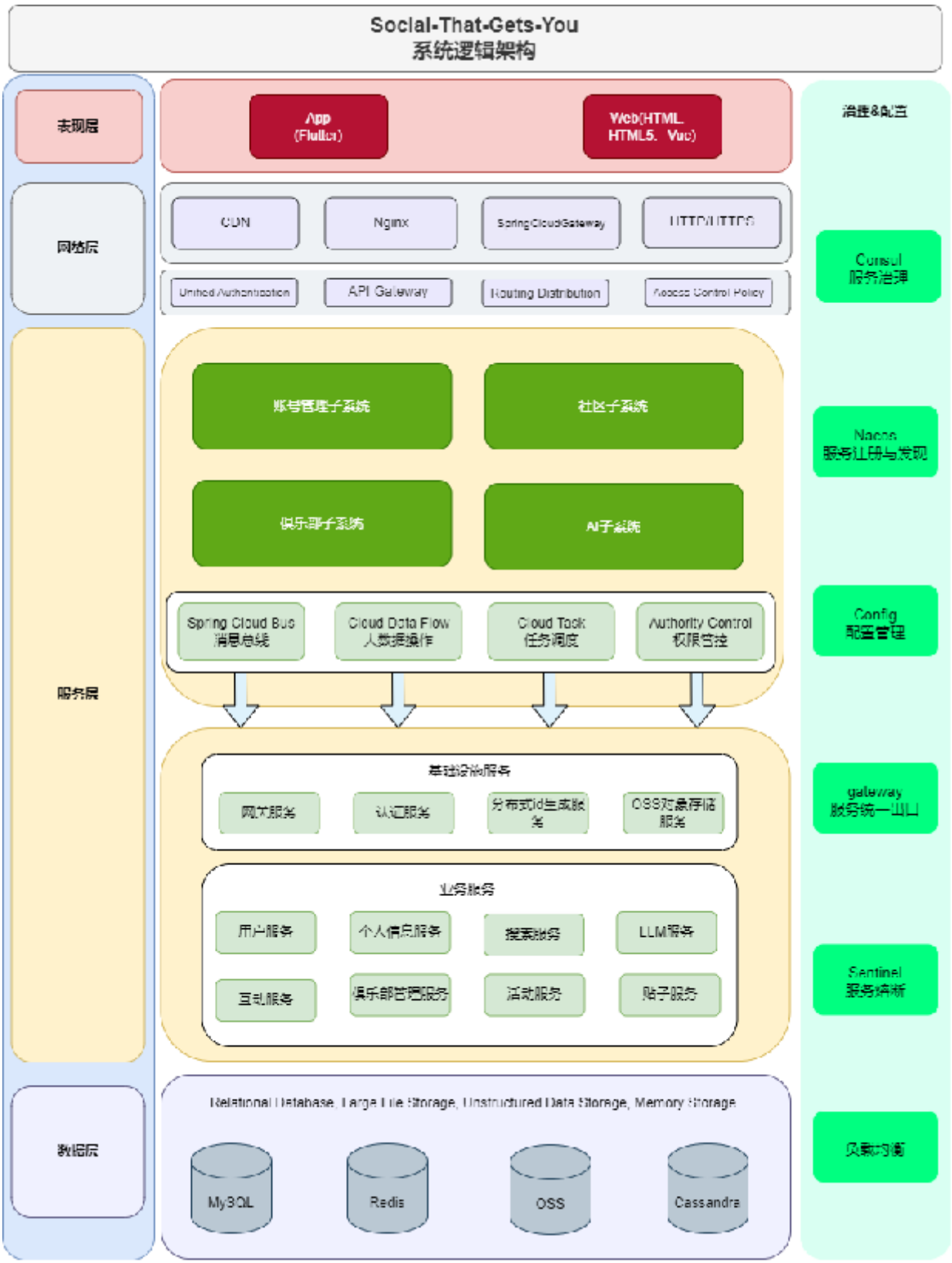
1.2.4 上下级系统

云计算平台和社媒开放平台属于上级系统，为本系统提供了必要的基础设施支持和社交媒体接口，使得系统能更好地服务于用户；大语言模型系统和第三方验证系统作为下级系统，前者提供了高级的语言处理能力，后者则增强了系统的安全性，两者共同作用以满足特定的功能需求。

1.2.5 同级系统

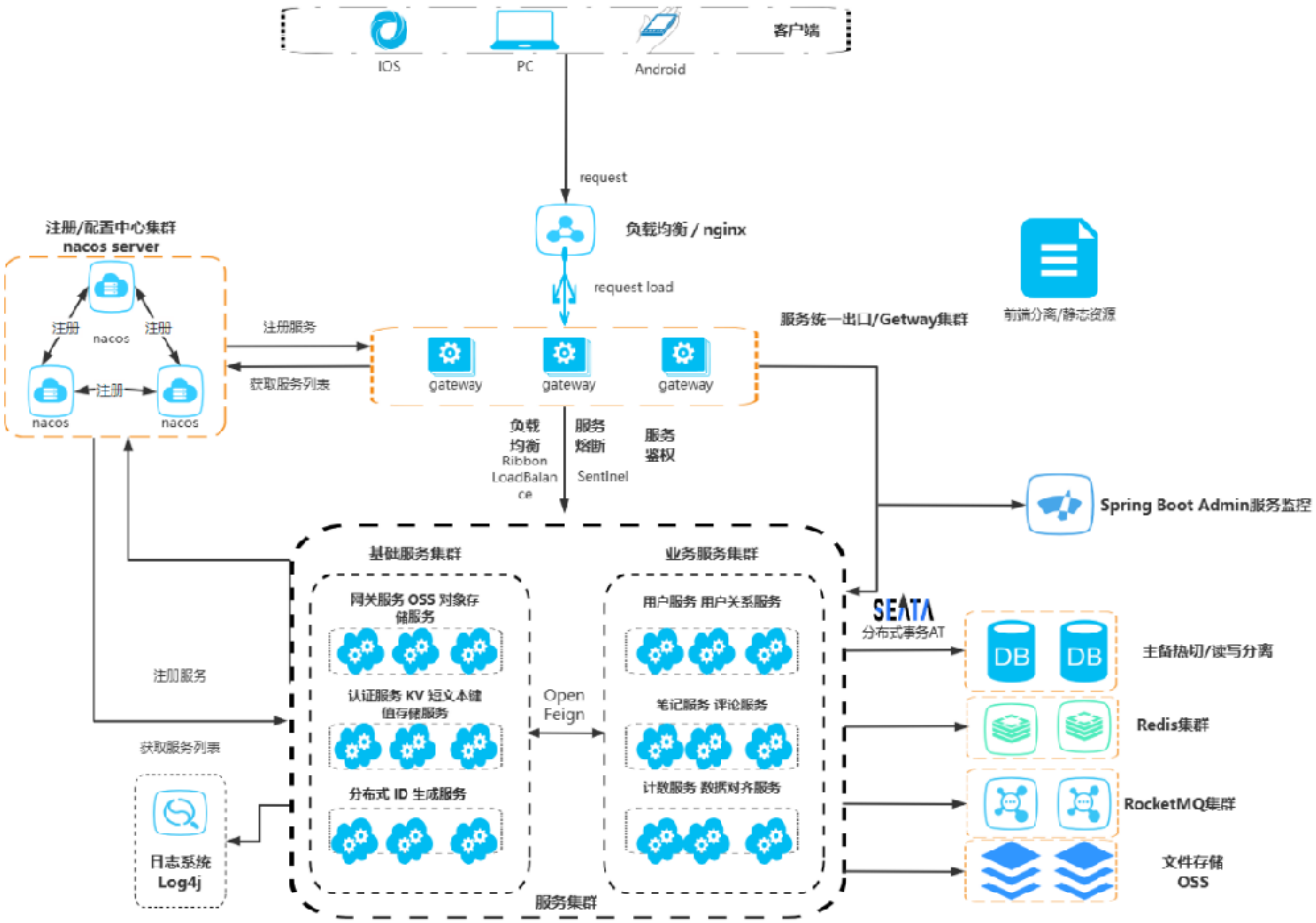
后台管理系统专为平台方设计，用于管理和监控整个平台的运行状态。该系统允许平台管理员执行多种操作，如用户管理、内容审核、数据分析等，确保平台的健康稳定运行。

1.3 系统架构设计



在结合技术选型与物理模型的设计后，我们完成了从分层架构到基于服务架构的过渡。在微服务架构选择上，采用了基于Spring Cloud Alibaba的微服务架构，以确保系统功能的完整性、数据安全性和高扩展性。微服务架构的优势在于能够将复杂系统拆解为多个独立且功能完备的服务，从而提高开发效

率和系统灵活性。系统架构层次分为网关层、基础设施层、业务服务层、中间件与基础设施层以及技术栈层。



系统包含客户端、网关集群、服务集群、注册中心、日志系统和分布式事务处理模块等核心组件。客户端通过网关集群发送请求，由 Nginx 进行负载均衡，网关负责统一入口和服务熔断。业务服务集群和基础服务集群通过 Open Feign 进行通信，依赖 Nacos 作为注册与配置中心。后端采用 Seata 实现分布式事务处理，支持 Redis 缓存、RocketMQ 消息队列、OSS 文件存储和主备分离数据库架构，同时配备 Spring Boot Admin 实现服务监控，保障高可用性和系统稳定性。

2. 组件设计

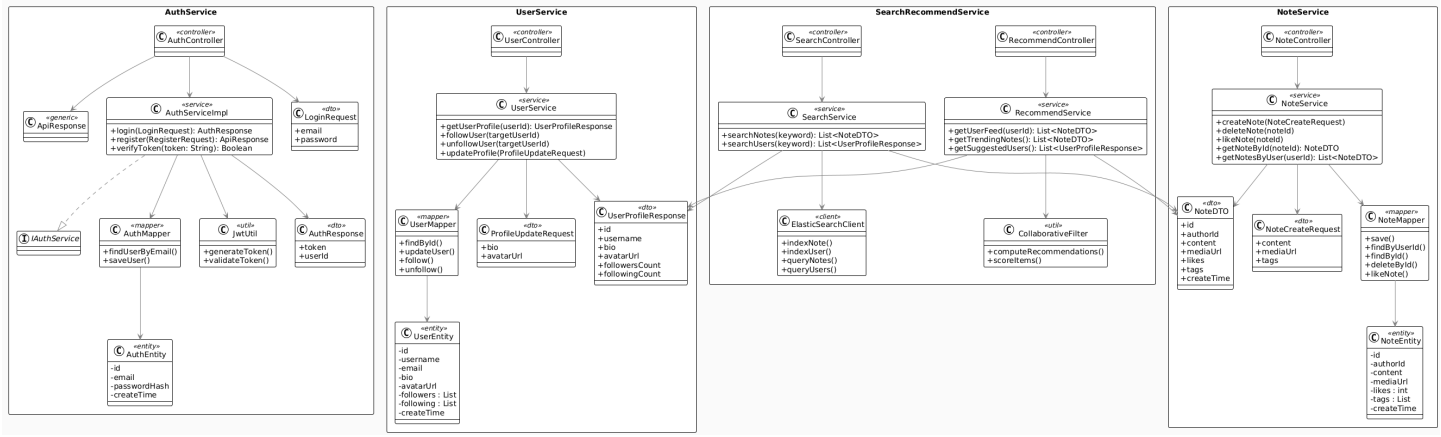
2.1 组件概要设计

2.1.1 组件功能概述

- 认证服务：负责用户登陆注册以及token验证
- 用户服务：负责用户管理个人信息，查看个人关注和粉丝
- 笔记服务：负责用户创建，编辑，删除笔记，还包括用户点赞和评论笔记
- 对象存储服务：负责存储用户上传的图片和视频等非结构化数据，使用阿里云OSS服务，确保用户数据安全

- 搜索和推荐服务：负责通过关键词或类别搜索，以及主页推荐功能

2.1.2 组件结构及其交互

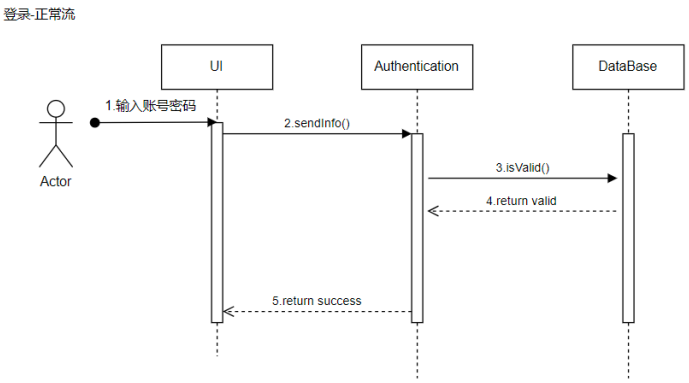


2.2 组件详细设计——时序图

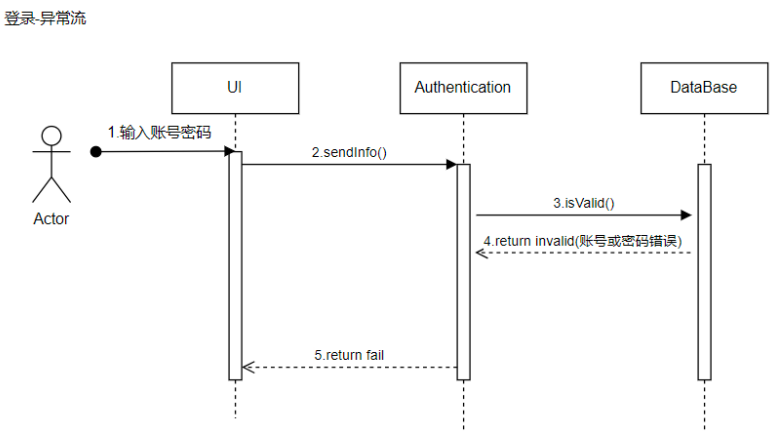
2.2.1 认证模块

2.2.1.1 用户登录

正常流



异常流

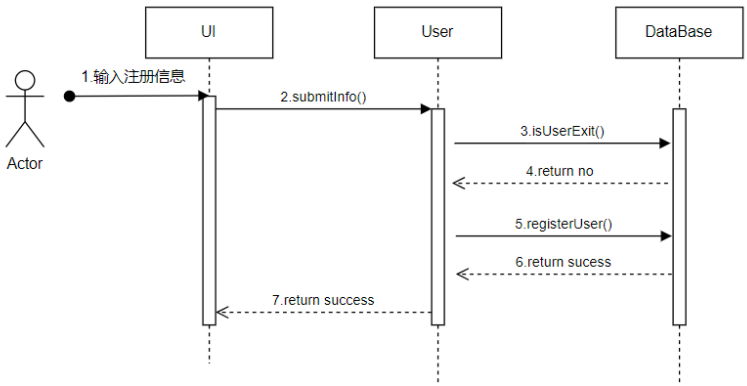


该序列图展示了用户登录的过程。用户在登录界面输入账号和密码后提交请求，系统接收到请求后调用验证模块检查用户的账号和密码是否匹配。若验证通过，系统记录用户的登录状态，并将用户引导

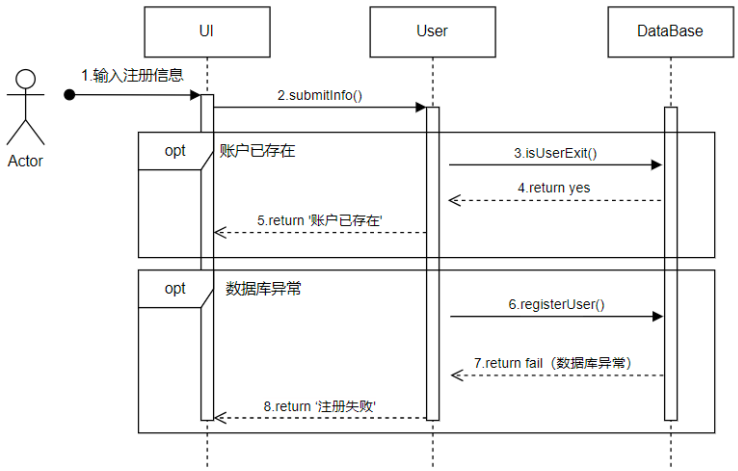
到主页面。若验证失败，系统向用户返回错误提示信息，用户可以根据提示重新输入正确的登录信息完成登录。

2.2.1.2 用户注册

正常流

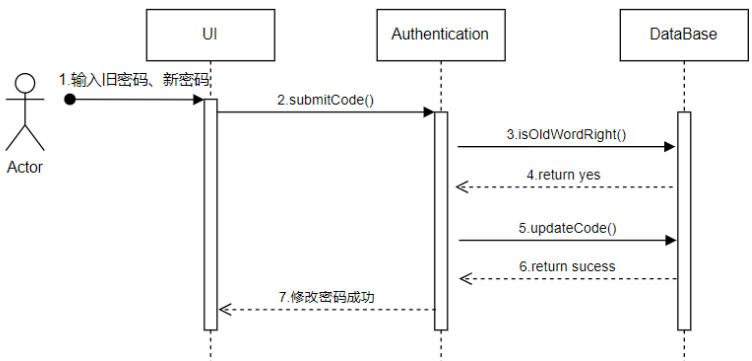


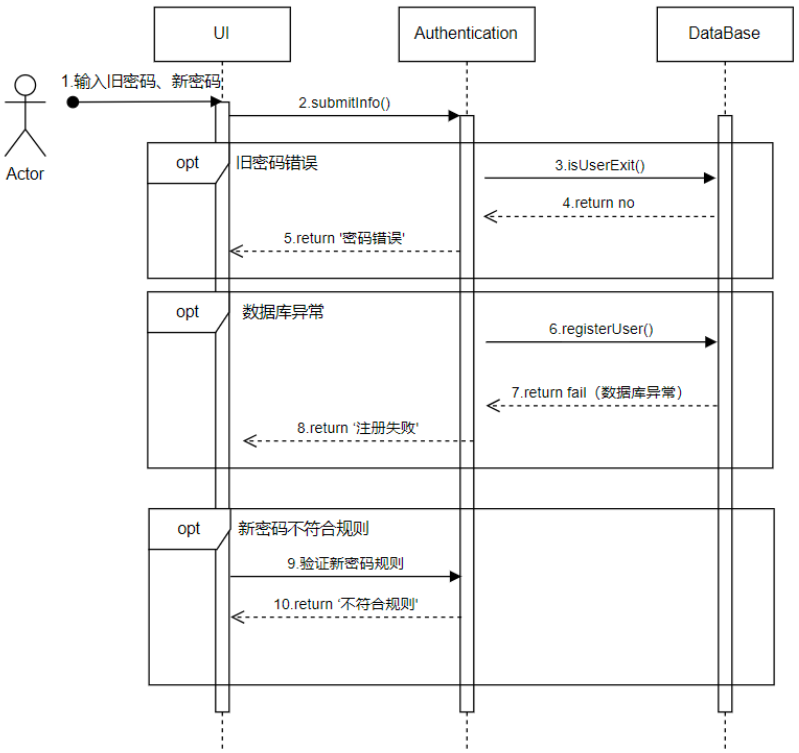
异常流



该序列图描述了用户注册账号的交互流程。用户在界面上输入注册信息并提交注册请求，系统首先验证用户输入的邮箱格式是否有效。若邮箱格式正确，系统生成验证码并通过邮件发送给用户。用户收到验证码后提交验证请求，系统对验证码进行校验，验证通过后将用户信息保存到数据库，并返回注册成功的消息给用户，完成注册流程。

2.2.1.3 密码修改

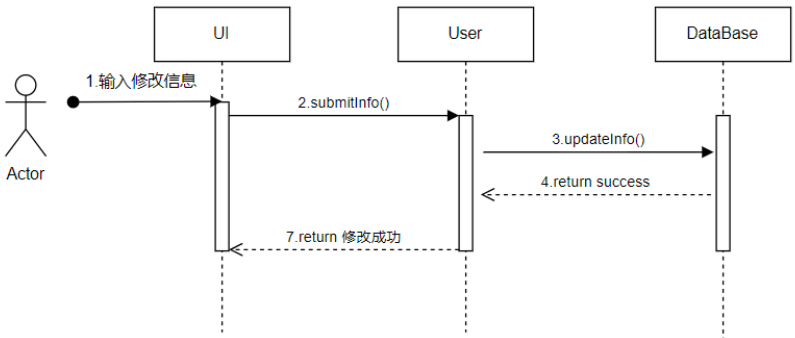


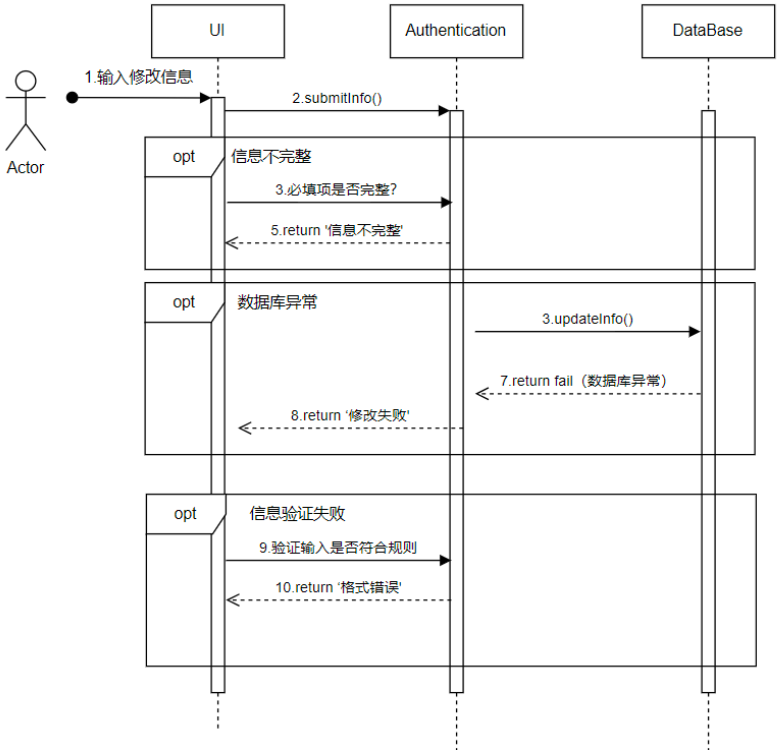


该序列图描述了用户在系统中修改密码的流程。用户首先访问密码修改界面，并提交旧密码和新密码。系统接收到请求后，调用验证模块检查旧密码是否正确，若验证通过，系统继续验证新密码的格式是否符合要求。当所有验证通过后，系统更新数据库中的用户密码信息，并向用户返回密码修改成功的消息，完成整个流程。

2.2.2 用户模块

2.2.2.1 个人信息查看与修改

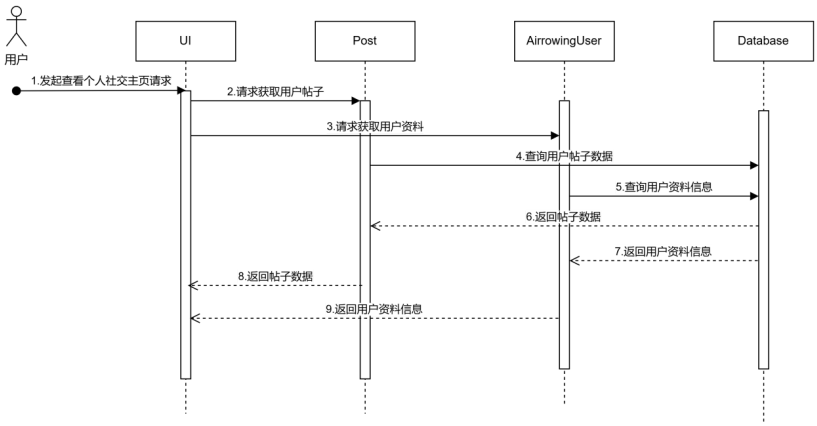




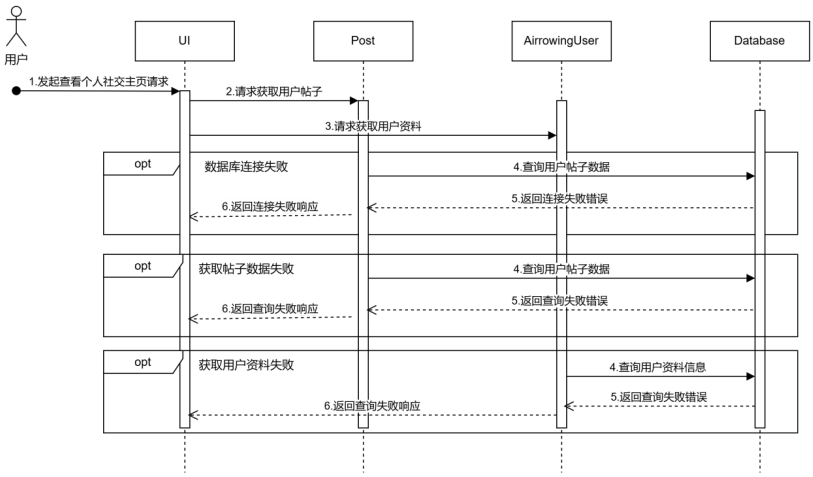
该序列图展示了用户修改个人信息的流程。用户通过界面发起请求查看当前的个人信息，系统从数据库中获取信息并展示给用户。用户修改个人信息后提交更改请求，系统验证提交的信息是否合法，验证通过后将修改后的信息更新到数据库。最后，系统向用户反馈修改成功的消息，从而完成个人信息的修改操作。

2.2.2.2 查看个人社交主页

正常流：



异常流：



参与者：

- **用户（User）**：系统的终端用户，发起查看个人社交主页的请求。
- **UI**：用户界面，负责接收用户输入并显示响应。
- **Post**：处理与帖子相关的业务逻辑实体，负责获取用户的帖子。
- **AirrowingUser**：处理与用户资料相关的业务逻辑实体，负责获取用户的资料信息。
- **Database**：存储数据的数据库系统，负责提供所需的数据。

正常流程:

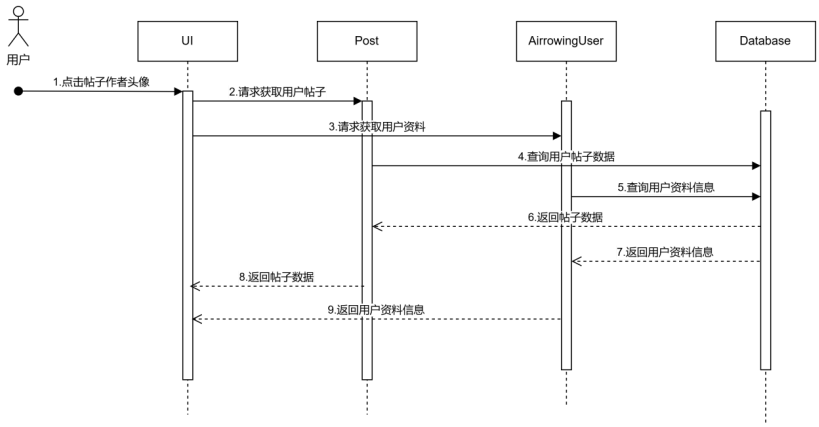
- 用户通过 **UI** 发起查看个人社交主页的请求。
- **UI** 将请求转发给 **Post** 和 **AirrowingUser**。
- **Post** 向 **Database** 请求获取用户的帖子数据。
- **AirrowingUser** 向 **Database** 请求获取用户的资料信息。
- **Database** 分别返回帖子数据和用户资料信息给 **Post** 和 **AirrowingUser**。
- **Post** 和 **AirrowingUser** 将获取的数据返回给 **UI**。
- **UI** 将个人社交主页的信息展示给用户。

异常流程：

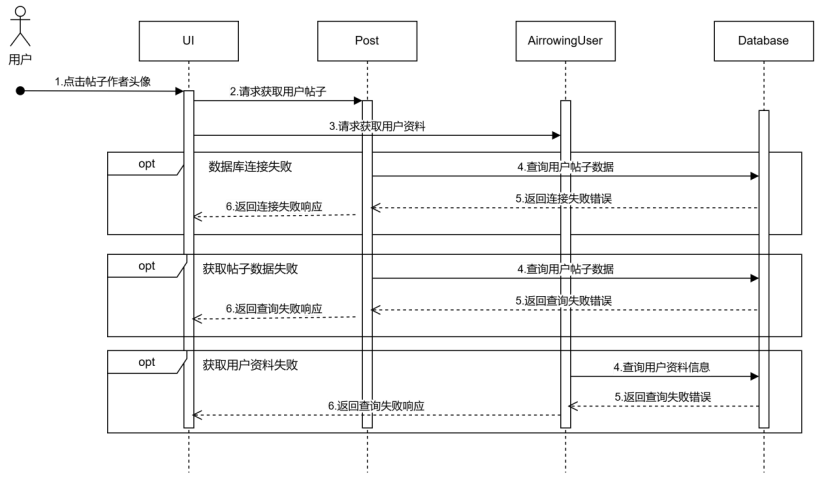
- **异常情况 1**：数据库连接失败。
- **异常情况 2**：获取帖子数据失败。
- **异常情况 3**：获取用户资料失败。

2.2.2.3 查看其他用户的社交主页

正常流：



异常流：



参与者：

- **用户（User）**：系统的终端用户，发起查看个人社交主页的请求。
- **UI**：用户界面，负责接收用户输入并显示响应。
- **Post**：处理与帖子相关的业务逻辑实体，负责获取用户的帖子。
- **AirrowingUser**：处理与用户资料相关的业务逻辑实体，负责获取用户的资料信息。
- **Database**：存储数据的数据库系统，负责提供所需的数据。

正常流程：

- 用户通过 **UI** 点击帖子作者的头像。
- **UI** 将请求转发给 **Post** 和 **AirrowingUser**。
- **Post** 向 **Database** 请求获取用户的帖子数据。
- **AirrowingUser** 向 **Database** 请求获取用户的资料信息。
- **Database** 分别返回帖子数据和用户资料信息给 **Post** 和 **AirrowingUser**。
- **Post** 和 **AirrowingUser** 将获取的数据返回给 **UI**。
- **UI** 将个人社交主页的信息展示给用户。

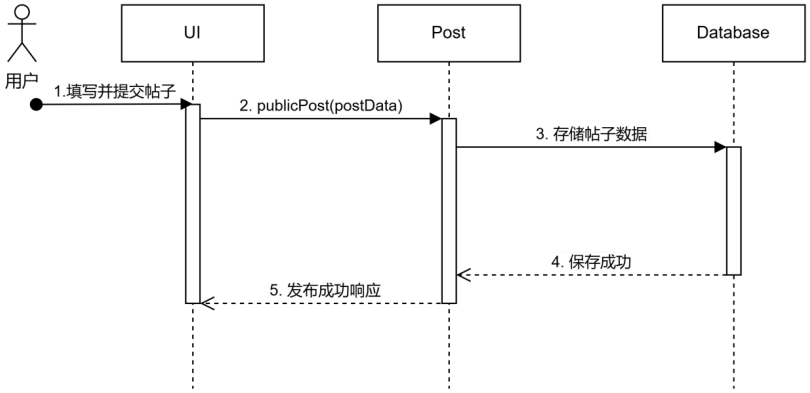
异常流程：

- **异常情况 1**：数据库连接失败。
- **异常情况 2**：获取帖子数据失败。
- **异常情况 3**：获取用户资料失败。

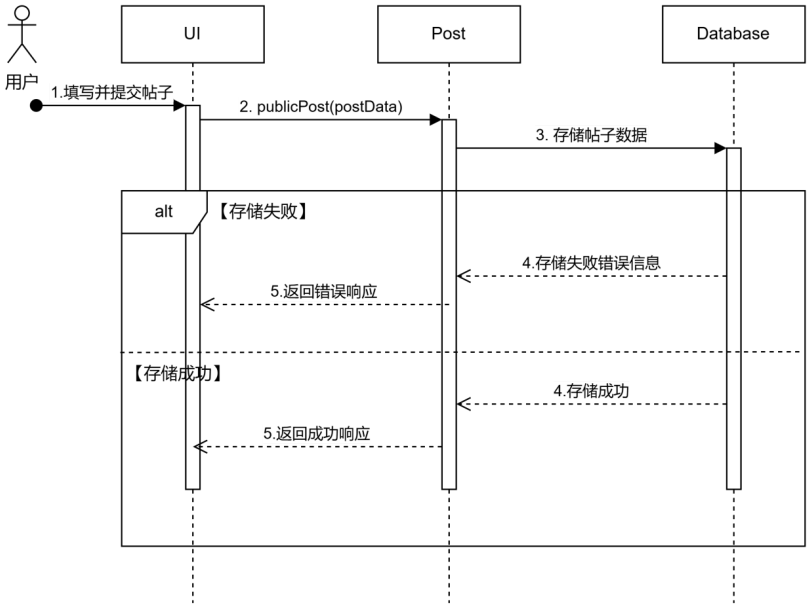
2.2.3 笔记模块

2.2.3.1 发布笔记

正常流：



异常流：



参与者：

- **用户（User）**：系统外部的终端用户，发起发布笔记请求。
- **UI**：用户界面，作为用户与系统交互的中介。
- **Post**：负责处理发布笔记的业务逻辑。
- **Database**：负责存储笔记数据的后端数据库系统。

正常流程：

- **步骤 1**：用户通过 **UI** 发起发布笔记请求。
- **步骤 2**：**UI** 将该请求转发给 **Post**。
- **步骤 3**：**Post** 处理该请求，并将笔记数据存储到 **Database**。
- **步骤 4**：**Database** 返回存储成功的确认。
- **步骤 5**：**Post** 将成功响应返回给 **UI**。

异常流程：

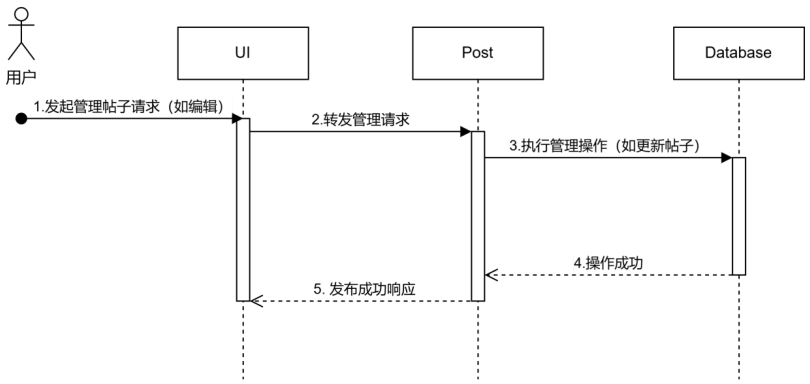
- **步骤 1-3**：与正常流程相同，用户通过 **UI** 发起请求，**UI** 转发给 **Post**，**Post** 尝试将数据存储到 **Database**。
- **步骤 4**：**Database** 返回存储失败的错误信息（如连接失败、数据验证错误等）。
- **步骤 5**：**Post** 捕获该错误，并生成错误响应。

条件分支：

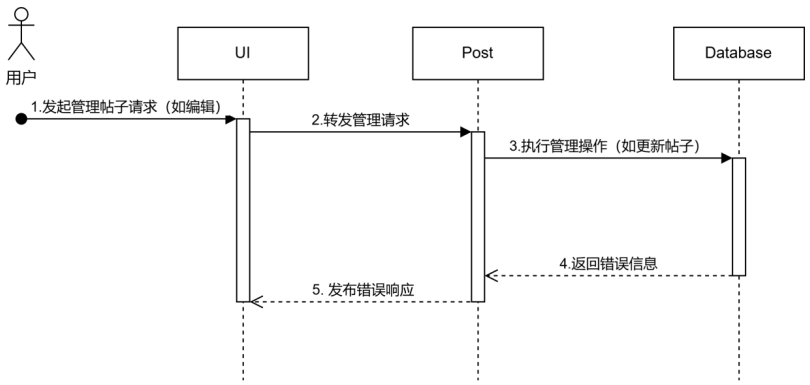
- 使用 `alt 存储失败` 分支来处理 **Database** 存储失败的情况。
- 使用 `else 存储成功` 分支处理存储成功的情况，以确保时序图涵盖所有可能的结果。

2.2.3.2 管理笔记

正常流：



异常流：



参与者：

- **用户（User）**：发起管理笔记的请求（如编辑、删除、查看）。

- **UI**：用户界面，负责接收用户输入并显示响应。
- **Post**：处理管理请求的实体，负责业务逻辑处理。
- **Database**：存储笔记数据的数据库系统。

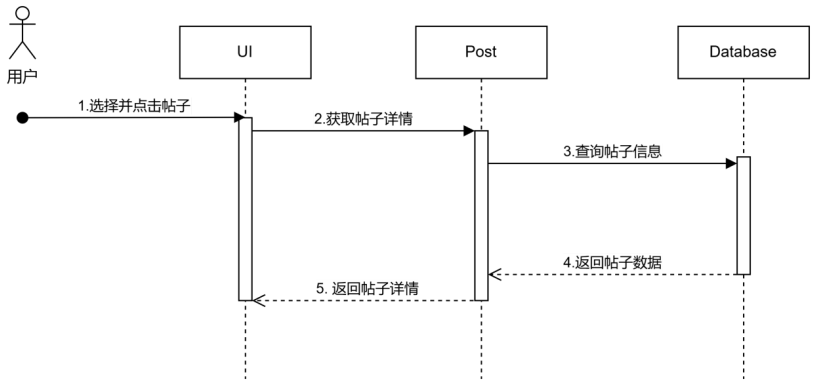
正常流程：

- 用户通过 **UI** 发起管理笔记请求（例如编辑笔记）。
- **UI** 将请求转发给 **Post**。
- **Post** 处理请求并与 **Database** 进行交互（如更新笔记数据）。
- **Database** 确认操作成功。
- **Post** 返回成功响应给 **UI**。
- **UI** 将成功信息展示给用户。

异常流程：

- 在正常流程的某个步骤中，如果发生异常（如数据库操作失败），则通过 `opt` 块表示异常处理流程。

2.2.3.3 浏览笔记



参与者：

- **用户（User）**：发起浏览笔记请求的终端用户。
- **UI**：用户界面，负责接收用户输入并显示响应。
- **Post**：处理浏览请求的实体，负责业务逻辑处理。
- **Database**：存储笔记数据的数据库系统。

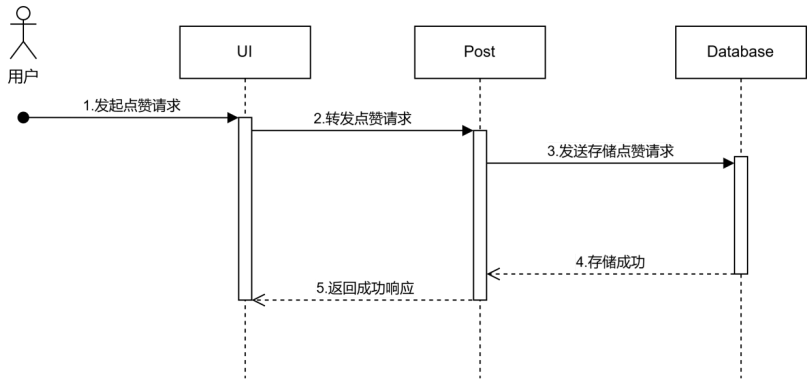
正常流程（Normal Flow）：

- 用户通过 **UI** 发起浏览笔记请求。
- **UI** 将请求转发给 **Post**。
- **Post** 处理请求并从 **Database** 获取笔记数据。
- **Database** 返回请求的数据。

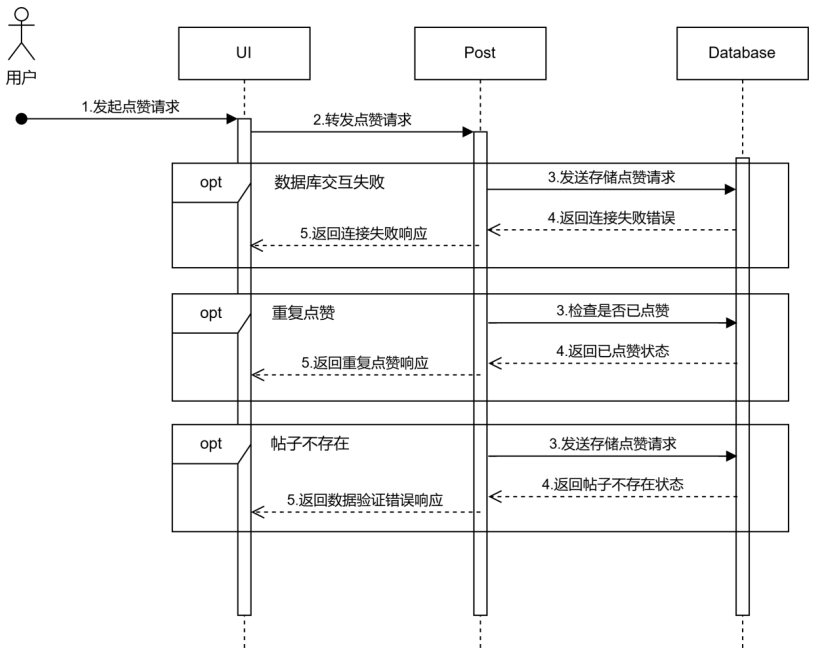
- **Post** 返回笔记数据给 **UI**。
- **UI** 将笔记数据显示给用户。

2.2.3.4 点赞笔记

正常流：



异常流：



参与者：

- **用户（User）**：发起点赞请求的终端用户。
- **UI**：用户界面，负责接收用户输入并显示响应。
- **Post**：处理点赞请求的实体，负责业务逻辑处理。
- **Database**：存储点赞数据的数据库系统。

正常流程：

- 用户通过 **UI** 发起点赞请求。
- **UI** 将请求转发给 **Post**。

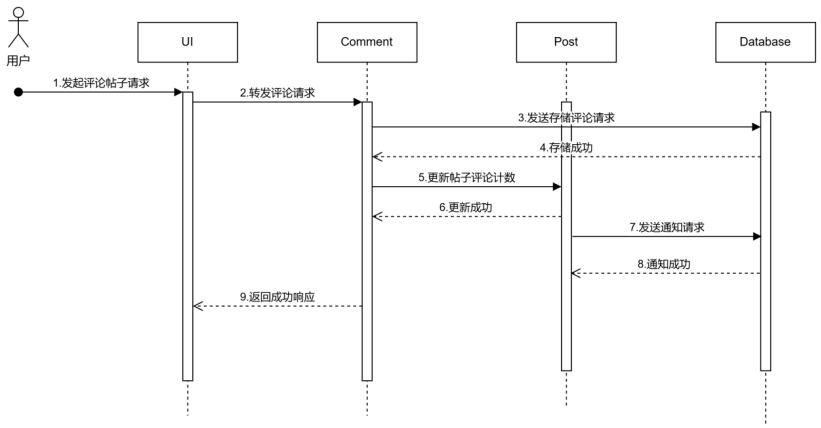
- **Post**处理请求并向 **Database** 发送存储请求。
- **Database** 返回存储成功的确认。
- **Post**返回成功响应给 **UI**。
- **UI** 将点赞成功的信息展示给用户。

异常流程：

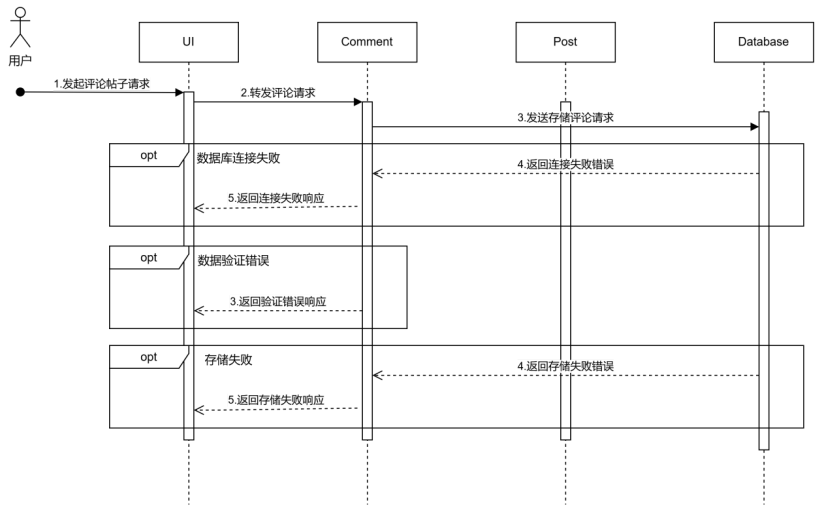
- **异常情况 1**：数据库连接失败。
- **异常情况 2**：数据验证错误（如帖子不存在）。
- **异常情况 3**：重复点赞（用户已点赞过该帖子）。

2.2.3.5 评论笔记

正常流：



异常流：



参与者：

- **用户（User）**：发起评论帖子请求的终端用户。
- **UI**：用户界面，负责接收用户输入并显示响应。
- **Comment**：处理评论请求的实体，负责业务逻辑处理。

- **Post**：处理与帖子相关的业务逻辑，如更新评论计数和通知作者。
- **Database**：存储评论数据的数据库系统。

正常流程：

- 用户通过 **UI** 发起评论帖子请求。
- **UI** 将请求转发给 **Comment**。
- **Comment** 处理请求并向 **Database** 发送存储请求。
- **Database** 返回存储成功的确认。
- **Comment** 通知 **Post** 更新评论计数。
- **Post** 更新评论计数并向 **Database** 发送通知请求。
- **Database** 返回通知成功的确认。
- **Comment** 返回成功响应给 **UI**。
- **UI** 将评论成功的信息展示给用户。

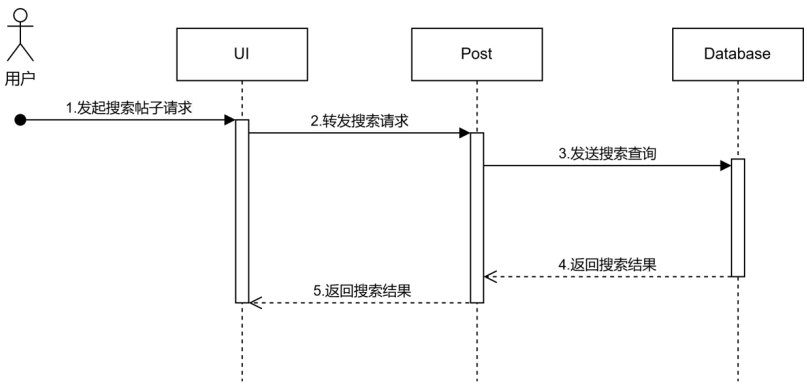
异常流程：

- **异常情况 1**：数据库连接失败。
- **异常情况 2**：数据验证错误（如评论内容为空或超长）。
- **异常情况 3**：存储失败（如数据库操作错误）。

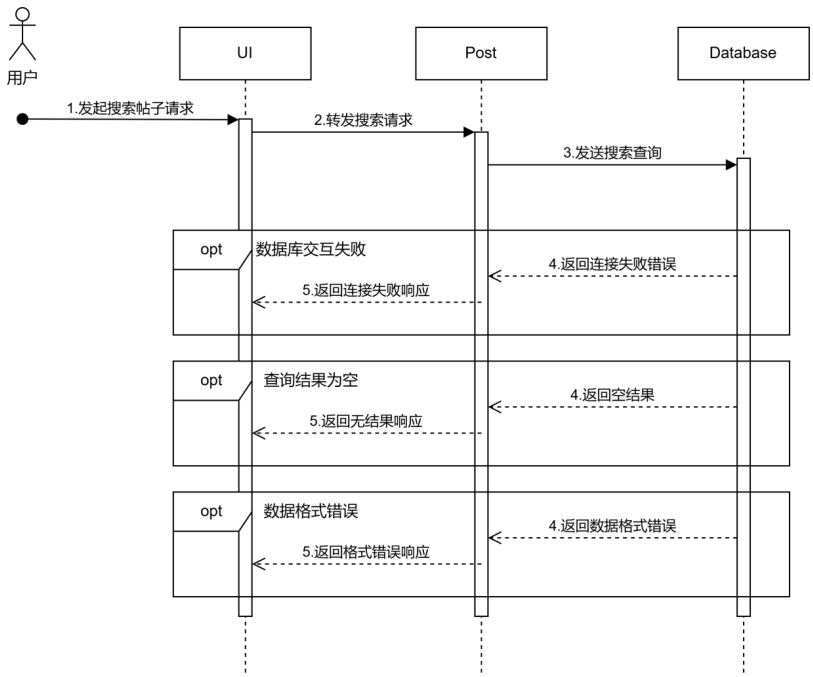
2.2.4 搜索模块

2.2.4.1 搜索笔记

正常流：



异常流：



参与者：

- **用户（User）**：发起搜索帖子请求的终端用户。
- **UI**：用户界面，负责接收用户输入并显示响应。
- **Post**：处理搜索请求的实体，负责业务逻辑处理。
- **Database**：存储帖子数据的数据库系统。

正常流程：

- 用户通过 **UI** 发起搜索帖子请求。
- **UI** 将请求转发给 **Post**。
- **Post** 处理请求并向 **Database** 发送搜索查询。
- **Database** 返回匹配的帖子数据。
- **Post** 返回搜索结果给 **UI**。
- **UI** 将搜索结果展示给用户。

异常流程：

- **异常情况 1**：数据库连接失败。
- **异常情况 2**：查询结果为空（无匹配帖子）。
- **异常情况 3**：数据格式错误导致查询失败。

3. UI设计

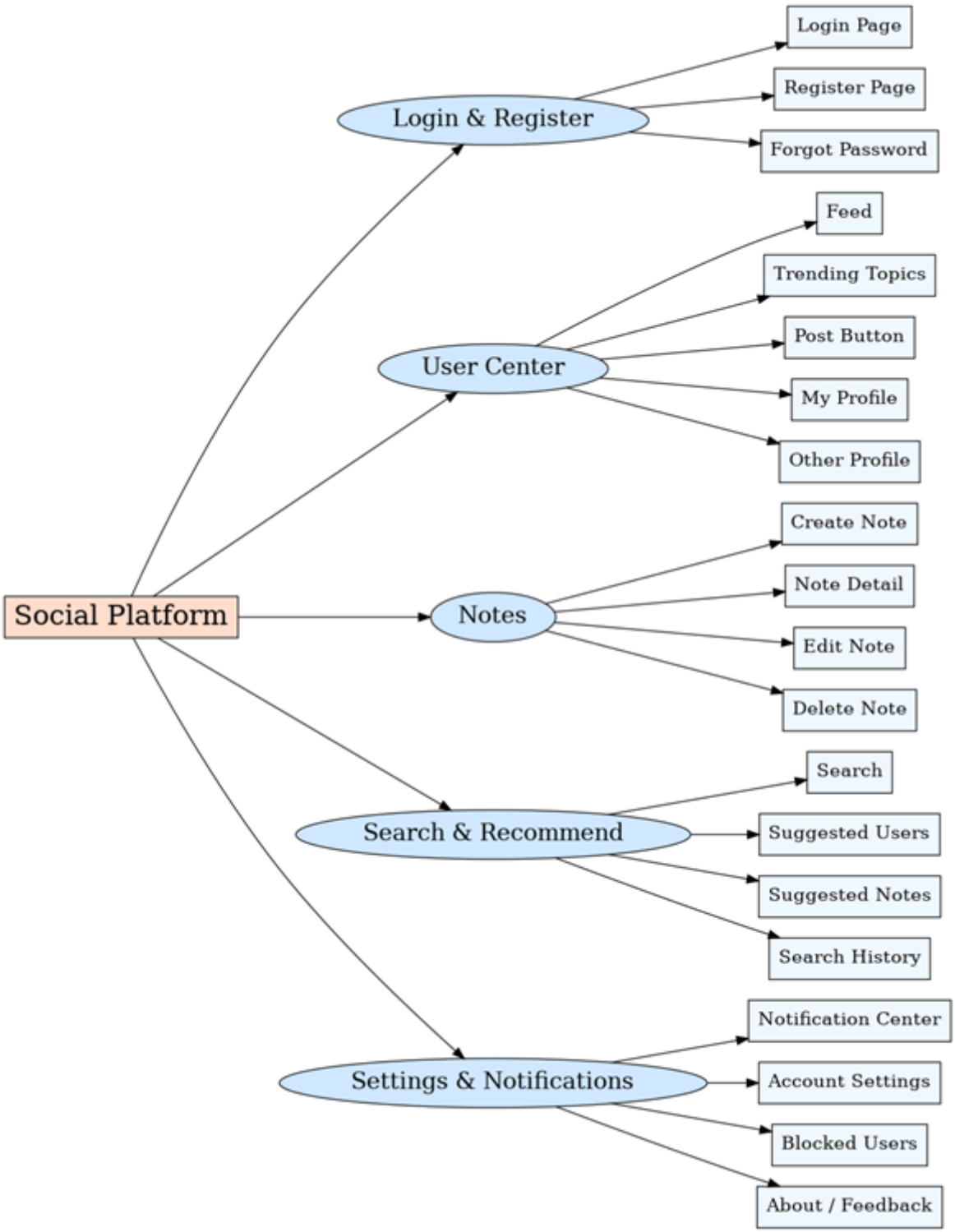
3.1 设计概览

a. 设计目标

- **用户友好**：通过直观的导航与简洁布局，降低用户学习门槛。
- **功能聚焦**：围绕“记录与分享”核心场景，优化高频操作路径（如笔记发布、互动）。
- **一致性**：统一视觉语言（色彩、图标、排版），强化品牌认知。

b. **工具与原型**

- **工具**：采用 **Axure** 完成高保真原型设计，支持动态交互与界面跳转。
- **覆盖范围**：设计 **10+核心界面**，涵盖登录、首页、笔记管理、个人中心等场景。



3.2 核心界面展示

3.2.1 登录与注册流程

- **双模式登录**：支持“验证码快捷登录”与“账号密码登录”，适配不同用户习惯。
- **自动化注册**：首次输入邮箱验证码即完成注册，简化流程。
- **交互细节**：
  - 输入框实时校验（如邮箱格式）。
  - 验证码发送后倒计时提示，避免重复请求。

3.2.2 首页导航与内容流

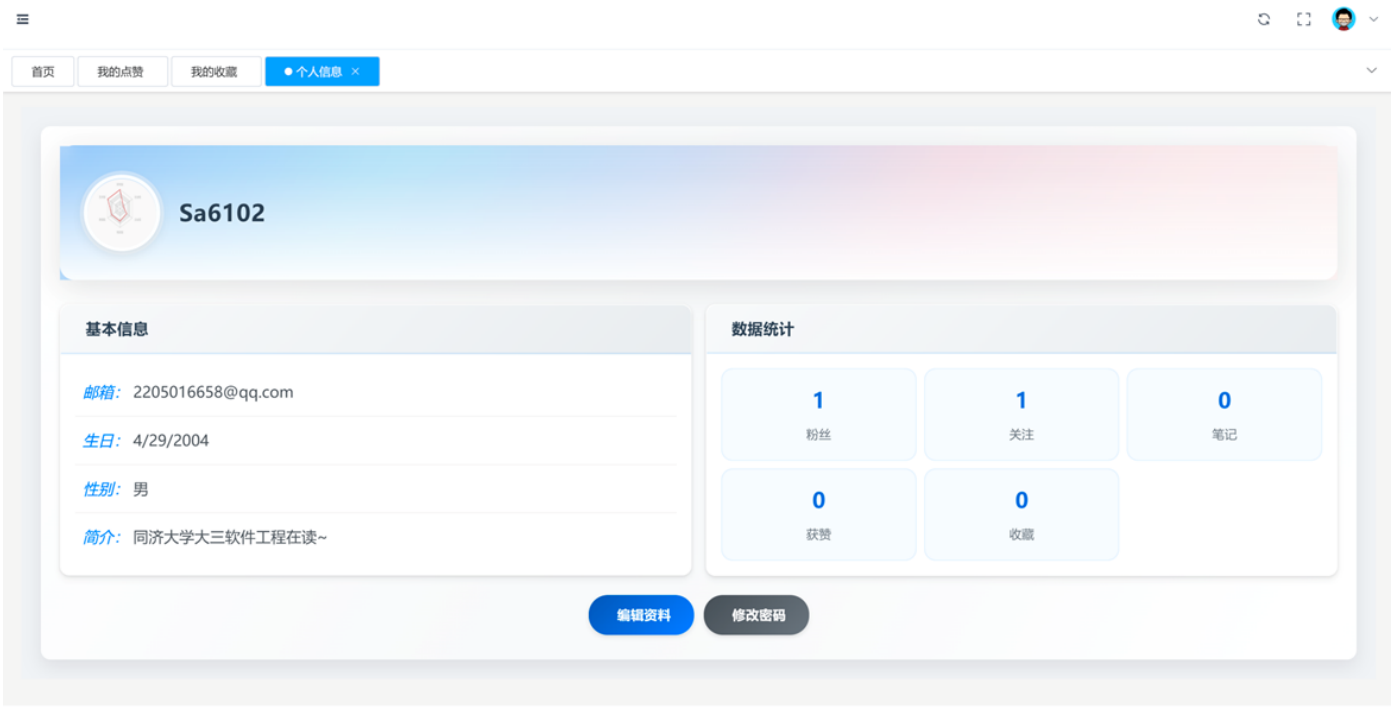
- **侧边栏导航**：
  - 五大核心模块：首页、我的笔记、收藏/点赞、关注/粉丝、个人信息。
  - 一键跳转，快速触达功能页面。
- **动态内容展示**：
  - 滚动浏览推荐内容，支持点击进入笔记详情页。
  - **置顶功能**：用户可手动将重要笔记置顶至主页顶部。

3.2.3 笔记发布与管理界面

- **发布功能**：
  - 图文混合编辑，支持图片上传与实时字数统计。
  - **可见范围设置**：提供“公开/仅好友/私密”三级权限控制。
- **管理功能**：
  - 笔记编辑：修改内容或调整可见范围。
  - 删除保护：二次确认避免误操作。

3.2.4 个人信息与互动管理

- **信息展示页**：
  - 集中显示头像、粉丝数、互动数据等核心信息。
  - 支持修改用户名、简介、头像等个人资料。
- **互动记录**：
  - **我的收藏**：按时间排序，支持取消收藏，内容默认私密。
  - **我的点赞**：历史点赞记录可追溯，支持快速取消。



3.3 交互与功能亮点

a. 高效导航设计

- **侧边栏与底部联动**：确保跨模块操作流畅，减少跳转层级。
- **即时反馈机制**：
  - 点赞/收藏后图标状态实时更新，增强操作感知。
  - 评论发布后自动刷新列表，提升互动体验。

b. 异常处理与提示

- **登录容错**：明确提示错误原因（如密码错误、网络问题）。

3.4 系统集成与安全设计

a. 与后端服务协同

- **认证服务**：通过统一接口实现登录状态管理，支持多设备安全登出。
- **笔记服务**：与评论、收藏功能深度集成，确保数据实时同步。

b. 隐私与安全机制

- **账户安全**：
  - 强制密码复杂度规则，支持定期修改密码。
  - 修改密码后自动登出其他设备，防止未授权访问。
- **内容隐私**：

- 笔记可见范围灵活控制，满足用户隐私需求。
- 收藏内容仅用户可见，保障私密性。

4. 接口设计

4.1 系统架构与协议规范

- **架构**：微服务架构，服务包括 `user`，`note`，`user-relation` 等。
- **通信协议**：HTTP RESTful 风格，JSON 作为数据交换格式。
- **安全认证**：所有接口需在 Header 中添加 `Authorization: Bearer <JWT_TOKEN>`。

4.2 全局定义

4.2.1 统一响应格式

```
{
  "code": 200,          // 状态码（200=成功，其他为错误码）
  "message": "success",  "data": {}           // 业务数据（可为空）
}
```

4.2.2 鉴权机制

- **JWT 鉴权**：敏感操作（如修改密码、删除笔记）需校验用户权限。
- **权限校验**：操作资源时需验证归属权（如删除笔记需校验作者身份）。

4.2.3 错误码体系

错误码	含义
400	参数校验失败
401	未授权
403	禁止访问（权限不足）
404	资源不存在
500	服务端内部错误

4.3 接口清单

4.3.1 用户服务（UserService）

接口功能	HTTP方法及路径	请求参数/体	响应体	描述
用户注册	POST /users	RegisterUserRequestDTO	Response<Long>	注册用户，返回用户ID
获取当前用户信息	GET /users/me	需 Authorization 头	Response<UserDetailResponseVO>	获取登录用户详细信息
修改密码	PUT /users/{userId}/password	UpdatePasswordRequestDTO	Response<?>	需验证原密码
通过邮箱查询用户	GET /users?email={email}	email（URL参数）	Response<UserByEmailResponseDTO>	根据邮箱查询用户信息

4.3.2 用户关系服务（UserRelationService）

接口功能	HTTP方法及路径	请求参数/体	响应体	描述
关注用户	POST /users/{userId}/follow	需 Authorization 头	Response<?>	关注目标用户
取消关注	DELETE /users/{userId}/follow	需 Authorization 头	Response<?>	取消关注目标用户
分页查询粉丝列表	GET /users/{userId}/fans	page=0&size=10（URL参数）	PageResponse<FansResponseVO>	分页获取用户的粉丝列表

4.3.3 笔记服务（NoteService）

接口功能	HTTP方法及路径	请求参数/体	响应体	描述
发布笔记	POST /notes	PublishNoteRequestVO	Response<Long>	返回新笔记ID
获取笔记详情	GET /notes/{noteId}	无	Response<NoteDetailResponseVO>	通过多级缓存查询详情
删除笔记	DELETE /notes/{noteId}	需校验操作者权限	Response<?>	清理缓存及相关计数
分页查询评论	GET /notes/{noteId}/comments	page=0&size=10（URL参数）	PageResponse<CommentVO>	分页获取笔记的一级评论

4.3.4 计数服务（CountService）

接口功能	HTTP方法及路径	请求参数/体	响应体	描述
获取笔记计数	GET /count/notes/{noteId}	无	Response<NoteCountDTO>	点赞数、评论数等
批量更新计数	POST /count/messages	List<CountMessageDTO>	Response<?>	异步处理计数消息

4.3.5 搜索服务（SearchService）

接口功能	HTTP方法及路径	请求参数/体	响应体	描述
语义化搜索笔记	POST /search/notes	SemanticSearchRequestVO	Response<List<SearchNoteVO>>	自然语言查询笔记
获取缓存结果	GET /search/cache	query（URL参数）	Response<List<SearchResultVO>>	读取缓存的搜索结果

4.3.6 推荐服务（RecommendService）

接口功能	HTTP方法及路径	请求参数/体	响应体	描述
个性化笔记推荐	GET /recommend/notes	userId（需鉴权）	Response<List<RecommendNoteVO>>	基于用户兴趣的推荐

4.3.7 AI模型服务（AIModelService）

接口功能	HTTP方法及路径	请求参数/体	响应体	描述
生成文本向量	POST /ai/embedding	{"content": "文本内容"}	Response<String>	用于语义搜索的向量嵌入
自动生成标签	POST /ai/tags	{"content": "文本内容"}	Response<List<String>>	为内容生成分类标签

4.4 DTO/VO字段说明（部分）

4.4.1 `RegisterUserRequestDTO`

字段名	类型	必填	描述
email	String	是	用户邮箱
password	String	是	密码（加密）
nickname	String	否	用户昵称

4.4.2 `NoteDetailResponseVO`

字段名	类型	描述
id	Long	笔记ID
title	String	标题
content	String	内容（HTML）
authorId	Long	作者ID

4.5 性能优化说明

4.5.1 多级缓存：

- `GET /notes/{noteId}` 接口使用本地缓存（5分钟）+ Redis缓存（30分钟）。

4.5.2 异步处理：

- 计数更新通过消息队列批量处理，保障高并发场景下的系统稳定性。

4.5.3 幂等性设计：

- 计数服务消费消息时，通过唯一ID去重，避免重复计数。

5. 数据设计

依据数据建模进行数据库逻辑设计和物理设计。指出本软件系统内部所使用的每个数据库结构的名称、标识符以及它们之中每个数据项、记录、文卷和系的标识、定义、长度以及它们之间的层次的或表格的相互关系。给出本系统内部所使用的每个数据库结构中的每个数据项的存储要求，访问方法、存取单位、存取的物理关系（索引、设备、存储区域）、设计考虑等。

5.1 数据库设计规范

5.1.1 数据库命名规范

采用26个英文字母(区分大小写)和0-9的自然数(经常不需要)加上下划线组成，命名简洁明确，多个单词用下划线''分隔，一个业务项目一个数据库，多个业务项目慎用同一个数据库。

5.1.2 表命名规范

1. 采用26个英文字母(区分大小写)和0-9的自然数(经常不需要)加上下划线'组成，命名简洁明确多个单词**用"下划线"分隔**。
2. **全部小写命名，禁止出现大写。**
3. 禁止使用数据库关键字，如:name，time，datetime，password等。
4. 表名称不应该取得太长(一般不超过三个英文单词)。
5. 表的名称一般使用名词或者动宾短语。
6. 用单数形式表示名称，例如，使用employee，而不是employees。
7. **明细表**的名称为:主表的名称+字符dtl(detail缩写)。例如:采购订单的名称为:po_order，则采购订单的明细表为:po_orderdtl。
8. 表必须填写描述信息(使用SQL语句建表时)。

5.1.3 表设计规范

1. 表必须定义主键，默认为ID，整型自增，如果不采用默认设计必须咨询DBA进行设计评估。
2. ID字段作为自增主键。一般所有表都要有id,id必为主键，类型为bigintunsigned，单表时自增、步长为1。一般情况下主键id和业务没关系的。
3. 建议不使用外键,数据的完整性靠程序来保证。
4. 多表中的相同列，必须保证列定义一致。
5. 国内表默认使用InnoDB，表字符集默认使用gbk，国际默认使用utf8的表。
6. 一般情况下每张表都有着四个字段create_id,create_time,update_id,update_time,其中create_id表示创建者id，create_time表示创建时间，update_id表示更新者id，update_time表示更新时间，这四个字段的作用是为了能够追踪数据的来源和修改。
7. 单表一到两年内数据量超过500w或数据容量超过10G考虑分表，且需要提前考虑历史数据迁移或应用自行删除历史数据。
8. 单条记录大小禁止超过8k，一方面字段不要太多，有的都能上百，甚至几百个，另一方面字段的内容不易过大，像文章内容等这种超长内容的需要单独存到另一张表。
9. 日志类数据不建议存储在 MSOL上，优先考虑Hbase或OB，如需要存储请找DBA评估使用压缩表存储。

5.1.4 字段命名规范

1. 采用26个英文字母(区分大小写)和0-9的自然数(经常不需要)加上下划线’ "组成，命名简洁明确,多个单词**用"下划线"分隔**。
2. **全部小写命名，禁止出现大写。**
3. 字段必须填写描述信息。

4. 禁止使用数据库关键字，如:name，time，datetime，password 等。
5. 字段名称一般采用名词或动宾短语。
6. 采用字段的名称必须是易于理解，一般不超过三个英文单词。
7. **在命名表的列时，不要重复表的名称**。例如，在名employe的表中避免使用名为employee_lastname的字段。
8. 不要在列的名称中包含数据类型。
9. 字段命名使用完整名称，禁止缩写。
10. 字段名命名示例
    - 使用下划线进行分隔，提升可读性。

      例如，user_name,user_id,is_friend,is_good。
    - 使用完整名称，提高可读性。

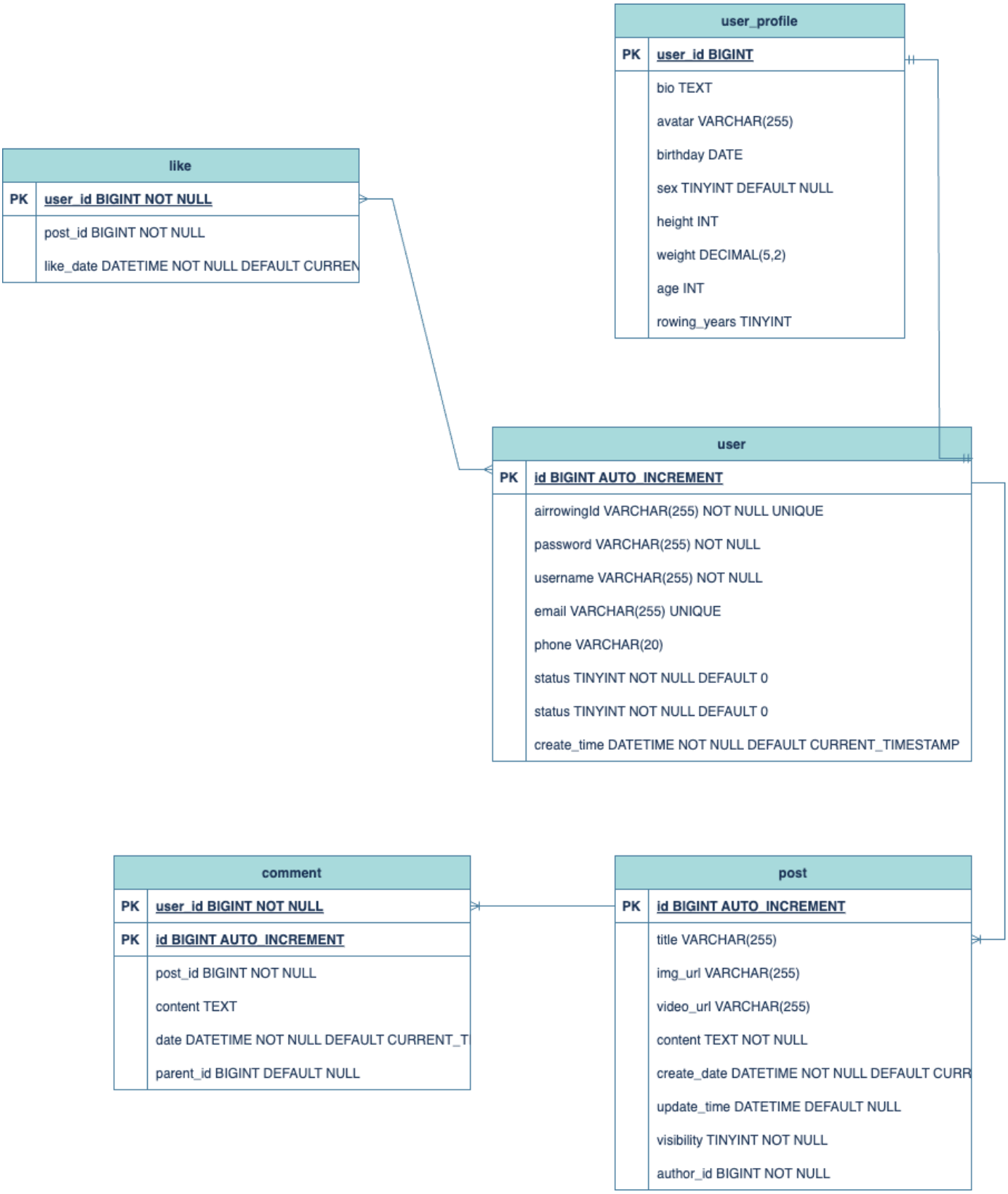
      例如，uid修改为user_id，pid修改为person_id。

5.1.5 字段设计

1. 表被索引列必须定义为not null，并设置default值。
2. 禁止使用float、double类型，建议使用decimal或者int替代。
3. 禁止使用blob、text类型保留大文本、文件、图片，建议使用其他方式存储，MySQL只保存指针信息。
4. 禁止使用varchar类型作为主键。

5.2 逻辑设计

5.2.1 总体ER图



5.2.2 实体间联系的介绍

user_profile 与 user

- **关系**：一对一

- **描述**：每个用户对应一个用户资料，用于展示个人社交主页信息。通过 `user_id` 字段与用户表关联。用户被删除时，其对应的资料也会被级联删除。

user 与 post

- **关系**：一对多
- **描述**：一个用户可以发布多个笔记，每个笔记由一个用户创建。

user 与 comment

- **关系**：一对多
- **描述**：一个用户可以发布多个评论，每条评论由一个用户创建。

post 与 comment

- **关系**：一对多
- **描述**：一个笔记可以包含多个评论，每条评论关联到一个笔记。

user 与 like

- **关系**：多对多
- **描述**：一个用户可以点赞多个笔记，一个笔记可以被多个用户点赞。通过 **like** 实体实现多对多关系。

5.3 物理设计

5.3.1 实体表

user：用户表

字段名	数据类型	长度	说明	备注
id	BIGINT		主键ID	主键，默认自增 NOT NULL AUTO_INCREMENT
airrowing_id	VARCHAR	255	用户的账号ID	唯一凭证
password	VARCHAR	255	用户密码	必填
username	VARCHAR	255	用户名	必填
email	VARCHAR	255	邮箱	唯一
phone	VARCHAR	20	手机号	可选
status	TINYINT		用户账号状态	0：启用，1：禁用

create_time	DATETIME		账号创建时间	默认当前时间

user_profile：用户资料表

字段名	数据类型	长度	说明	备注
user_id	BIGINT		与 `user.id` 关联，用户ID	外键，NOT NULL
bio	TEXT		用户的简介	可选
avatar	VARCHAR	255	用户头像文件路径	可选
birthday	DATE		用户生日	可选
sex	TINYINT		性别 (0：女 1：男)	默认值为 NULL
height	INT	(3)	用户身高（单位：厘米）	可选
weight	DECIMAL	(5,2)	用户体重（单位：千克，支持小数）	可选
age	INT		用户年龄	可选
rowing_years	TINYINT		用户赛艇经验年数（整数）	可选

post：笔记表

字段名	数据类型	长度	说明	备注
id	BIGINT		笔记的唯一标识符。	主键，自增 NOT NULL AUTO_INCREMENT
title	VARCHAR	255	笔记的标题。	可以根据需求调整长度
img_url	VARCHAR	255	笔记中包含的图片的URL。	
video_url	VARCHAR	255	笔记中包含的视频的URL。	
content	TEXT		笔记的正文内容。	NOT NULL
create_date	DATETIME		笔记创建的日期和时间。	默认当前时间，NOT NULL
update_time	DATETIME		笔记更新的日期和时间。	
visibility	TINYINT		笔记的可见性设置（如0表示私密，1表示公开）。	NOT NULL

author_id	BIGINT		发布笔记的用户ID，引用 **user** 实体。	外键，NOT NULL

comment：评论表

字段名	数据类型	长度	说明	备注
id	BIGINT		评论的唯一标识符。	主键，自增 NOT NULL AUTO_INCREMENT
user_id	BIGINT		外键，发布评论的用户ID，引用 **user** 实体。	外键，NOT NULL
post_id	BIGINT		外键，评论所关联的笔记ID，引用 **post** 实体。	外键，NOT NULL
content	TEXT		评论的内容。	
date	DATETIME		评论发表的日期和时间。	默认当前时间，NOT NULL
parent_id	BIGINT		父评论标识符，用来处理评论的回复。	可以自关联到 `comment(id)`

5.3.2 关系表

like：点赞表

字段名	数据类型	长度	说明	备注
post_id	BIGINT		点赞的笔记ID，引用 **post** 实体。	外键，NOT NULL
user_id	BIGINT		点赞的用户ID，引用 **user** 实体。	外键，NOT NULL
like_date	DATETIME		点赞的日期和时间。	默认当前时间，NOT NULL

5.3.3 性能优化

在本项目的数据库设计中，通过以下策略实现了对性能的初步优化，既保证了数据的完整性与一致性，也兼顾了后续可能的读写压力与查询效率：

1. **主键与自增列的使用**
   - 各个核心表（如 `user`，`post`，等）都使用了自增型主键（`BIGINT AUTO_INCREMENT`）来保证主键的唯一性，同时避免了不连续、无规律的业务型主键导致的查询效率下降。
   - 某些中间关联表（如 `like`，`comment`）以组合键或外键 + 自增主键的形式进行标识，进一步保证了数据的唯一性。
2. **合理的数据类型选择**
   - 针对较小范围的整型字段（如性别、年龄、状态等），使用 `TINYINT` 或者小数位数有限的 `DECIMAL`，节约存储空间的同时提高数据处理效率。
   - 对于时间字段，使用 `DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP` 等方式记录插入或更新时间，既能满足业务场景下对精确时间的需求，也便于后续对时间维度的筛选或排序。
   - 对文本量较大的字段（如 `post` 表与 `comment` 表的 `content` 列）使用 `TEXT`，而对于长度相对可控的描述、名称类字段（如 `name`，`title`）则使用 `VARCHAR`，实现灵活性与存储效率的平衡。
3. **适度的表拆分与结构设计**
   - 将用户的基础信息与附加信息分为 `user` 与 `user_profile` 两张表，避免单表字段过多、数据过大，提升查询效率及可维护性。
   - 针对笔记（`post`）、评论（`comment`）、点赞（`like`）等高频交互数据分别进行独立的表设计，既便于针对性地建立索引，也利于后续在高并发场景下进行分库分表或读写分离。
4. **外键与索引的配合使用**
   - 在涉及关联查询的表（如 `comment` 与 `post`、`like` 与 `user`）均通过外键或关联字段实现数据的一致性，同时也可基于外键列创建索引来提升关联查询性能。
   - 对典型的查询条件和排序字段（如 `post` 表中的 `author_id`，`create_date`，`comment` 表中的 `post_id`，`user_id` 等）建立合适的单列或组合索引，尽可能减少全表扫描，提高读性能。
5. **约束与唯一性**
   - 设计中对关键字段（如 `email`，`phone` 等）设置了唯一约束（`UNIQUE`），对状态与类型字段定义了合适的限制（`ENUM`、`TINYINT` 等），不仅确保数据质量，也有助于数据库层面进行快速判断与检索。
   - 通过 `NOT NULL` 与合适的默认值设置（如 `DEFAULT CURRENT_TIMESTAMP`、`DEFAULT 0` 等），降低了数据插入或更新时的处理开销，同时减少空值判断逻辑对系统性能的影响。