**《移动互联系统分析与设计》**

|  |  |
| --- | --- |
| **结课设计** |  |
| **一、系统需求文档** |  |

**(一)项目可行性分析与描述：**

**1.系统领域**

软件系统Snapnote是一个支持用户的自由创作和分享如笔记、图片、视频等，同时通过点赞、评论、关 注等社交功能促进用户之间的互动的系统。能够满足用户在学习、创作和社交方面的多重需求，帮助用 户记录生活、分享创意、建立社交联系，属于社交媒体和内容分享领域。随着用户数量的快速增长，平 台需要具备高并发处理能力和数据一致性，以确保用户操作的实时响应和系统的稳定运行。

**2.用户群体**

Snapnote用户群体广泛，适用于那些希望记录和分享个人创作、生活记录、学习笔记等内容的创作者， 满足了他们在内容创作和传播方面的需求；同时，系统也吸引了关注他人动态、参与互动的社交用户， 通过社交功能促进用户间的联系与互动。此外，平台还适用于那些希望分享和获取知识、经验的学习

者，提供了一个丰富的知识共享和交流环境。对于普通用户来说，系统通过个性化设置和社交功能，提 供了一个综合的内容管理和社交体验，帮助用户更好地管理自己的信息和与他人的关系。

系统通过一系列功能满足了不同用户群体的需求。在**内容创作**方面，通过“**我的笔记**”功能，用户可以记录 和分享个人创作、文章或学习笔记，满足了他们对内容创作和表达的需求。系统提供“**笔记详情**”功能使得 用户可以查看和编辑个人笔记，进一步优化内容展示和管理。

在**社交互动**方面，系统通过“**关注粉丝**”功能帮助用户与其他人建立联系，用户可以关注感兴趣的内容创作 者或社交圈子，保持与他人动态的实时互动。与此同时， **点赞**和**评论**等社交功能，让用户能够直接与他 人进行反馈和交流，增强了社交互动的深度与活跃度。通过“**我的点赞**”功能，用户可以查看自己收到的点 赞和评论，获得社交反馈并增强自己的社交存在感。系统的“**我的收藏**”功能还允许用户收藏自己喜欢的内 容，帮助他们高效管理和归档有价值的信息。

此外，**个性化管理**是系统的另一大亮点。通过“**个人信息**”功能，用户可以自由地修改和优化个人资料，满 足了用户在隐私保护和自我展示方面的需求。用户还可以根据自己的兴趣调整系统的展示内容，使平台 更加符合个人偏好，提供个性化的使用体验。这些功能帮助用户优化了社交和内容管理的方式，提高了 用户的满意度和平台粘性。

在内容管理方面，系统通过动态路由（如“**笔记详情**”的路径包含动态参数 :id 和 :userId）提供了灵活的数 据展示和内容管理方式。这使得用户能够根据自己的需求快速访问特定内容，系统自动根据动态参数展 示不同用户的笔记详情或其他内容，提升了平台在内容管理和个性化展示方面的能力。

总之，这个软件系统解决了用户在**内容创作、社交互动、个性化设置和内容管理**方面的多重需求，提供 了一个功能丰富、易于使用的平台。它不仅帮助用户记录和分享个人创作，还通过社交互动增强了用户 之间的联系，促进了平台的社区建设。同时，通过个性化管理和动态内容展示，系统优化了用户的内容 查看和管理体验，满足了不同用户群体在社交、创作、学习和管理方面的复杂需求。

**3.竞品比较**

市场上存在多款类似的图文发布交流分享平台如小红书， Evernote ， Notion等，相比于这些系统，我们 的Snapnote更加轻量化与专注性。专注于笔记分享和管理功能，界面简洁直观，在设计上追求清新简洁 的风格，可以让用户快速上手。用户不需要复杂的设置，使用门槛更低。

同时我们提高了个性化和互动性，提供置顶、隐私设置等灵活功能，满足不同用户的需求。允许用户个 性化展示笔记列表，支持分享和互动功能（如点赞、评论）。系统倾向于特定场景优化，打造一个内容 分享社区，更适合小众或垂直领域的用户。

但我们系统的用户基础较弱。与小红书、印象笔记等成熟平台相比，我们的用户量起步较低，吸引用户 初期可能较为困难。初期版本的功能相对基础，例如缺乏完善的推荐系统、笔记协作功能或 AI 智能辅助 等。品牌认知度较低：市场上已存在众多竞品，对于如何在用户中形成品牌认知和信任度是一个挑战。

**（二）系统功能**

**1.功能分解层次图 Snapnote系统**

卜── 用户管理

│ 卜── 添加用户信息 │ 卜── 删除用户信息 │ 卜── 修改用户信息 │ L── 用户信息查看 卜── 用户认证

│ 卜── 验证码登录 │ L── 账号密码登录

卜── 社交互动

│ 卜── 关注/取消关注 │ │ L── 关注列表查看 │ 卜── 点赞/取消点赞 │ │ L── 点赞列表查看 │ 卜── 评论

│ │ 卜── 发表评论

│ │ 卜── 删除自己的评论

│ │ 卜── 评论的评论（多层讨论） │ │ L── 评论列表查看

│ L── 笔记

│ 卜── 创建笔记

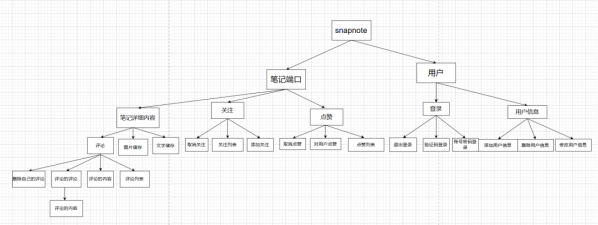
│ 卜── 笔记详细内容编辑

│ 卜── 笔记端口管理 │ L── 图片储存

卜── 数据管理和存储 │ 卜── 文字储存

│ L── 图片储存

L── 系统管理 卜── 退出登录 L── 修改密码



**用户管理**提供了对用户信息的全面控制。管理员或用户自己可以通过添加用户信息来创建新账户， 确保每位用户都能获得个性化的服务；删除用户信息功能则允许安全地移除不再需要的账户资料； 修改用户信息使用户能够更新个人资料，如联系方式、头像等，以保持资料的最新状态；而用户信 息查看功能则为用户提供了一站式的服务，让他们可以随时回顾和检查自己的账户详情，确保信息 的透明性和准确性。

**用户认证**模块保障了系统的安全性。它支持两种主要的登录方式：验证码登录为那些可能忘记密码 或者偏好更简便登录流程的用户提供了方便；账号密码登录则为追求传统安全性的用户保留了经典 的验证方法。这两种登录机制相辅相成，既满足了不同用户的需求，又确保了系统的访问安全。

**社交互动**模块极大地丰富了用户的在线体验。在这个模块中，用户不仅可以关注感兴趣的人或话

题，并通过关注列表查看已关注的对象动态，还可以取消关注。点赞列表记录了用户的喜好，帮助 他们追踪曾经赞赏过的精彩瞬间。用户还可以发表评论、删除自己的评论、参与多层讨论（即“评 论的评论” ），形成丰富的讨论链，促进思想交流。此外，笔记功能允许用户创建个性化笔记，编 辑详细内容，管理笔记端口，并储存相关图片，使得信息分享和保存更加多样化。

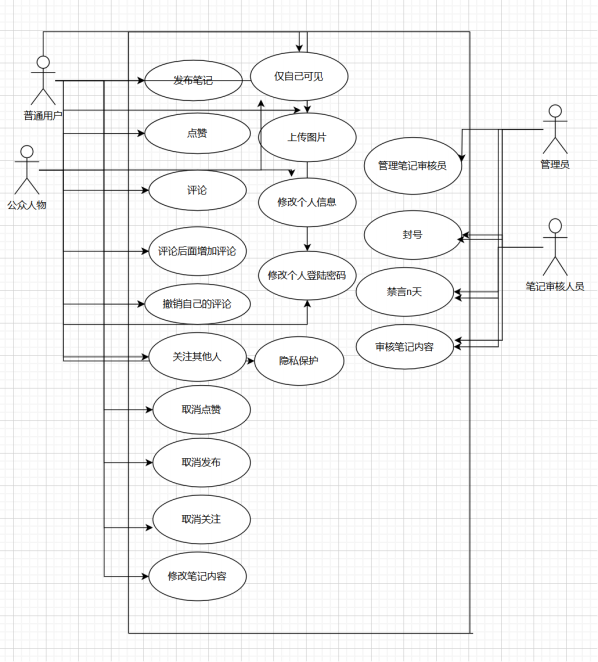
**数据管理和存储**模块确保了所有用户生成的内容都得到了妥善处理。文字储存负责保存文本形式的 信息，无论是评论还是笔记内容，都经过优化存储，以便快速检索和高效利用。图片储存则专门针 对视觉内容，保证每一张上传的图片都能得到高质量的保存，为用户提供优质的视觉体验。

最后，**系统管理**模块增强了系统的易用性。退出登录功能让用户每次使用后都能安全离开，支持修 改密码功能，保护个人隐私。

**2.UML用例图**

展示用户类型及存取权限分配方案

系统根据不同的用户角色分配了多种功能和权限，确保各类用户在平台上有不同的操作能力，同时保证 平台的安全与有序运行。



普通用户是平台的基础用户，能够发布笔记、点赞其他用户的笔记、发表评论，并将笔记设置为“仅自己 可见” ，以确保某些内容的隐私性。用户还可以上传图片，提升笔记的表现力。普通用户有修改个人信息 的权限，例如更新个人资料或密码等。同时，他们可以关注其他用户，查看他们的笔记内容。如果用户 觉得不再感兴趣，还可以取消关注，也可以取消已点赞的内容。用户在隐私保护方面有一定控制权，可 以调整哪些信息对外公开，哪些保持私密。

公众人物在平台上的角色类似普通用户，但可能会有更多的曝光和互动机会。虽然系统没有显式给出额 外的功能，他们的笔记、评论等内容通常会受到更广泛的关注，因此在某些情况下，公众人物可能需要 面对更高的审核和管理要求。

管理员拥有最强的权限，他们负责对平台上的用户进行管理。管理员可以封号，禁止某些用户在违反平 台规定时继续使用平台。管理员还可以禁言用户一段时间，限制他们在平台上的互动与评论。管理员的 权限还包括修改用户笔记的内容，取消发布的笔记，确保平台内容的合规性。

笔记审核员主要负责审核平台上的笔记内容，确保发布的笔记符合平台的规定。审核员可以查看并审核 用户发布的笔记，如果内容不符合标准，可以要求修改或直接删除。审核员还可以在评论后进行附加评 论，以解释审核决定，并且有权限撤销自己发表的评论。

**3.客户端应用与服务端实现功能**

Snapnote前后端分离，使用Gateway进行统一路由管理和负载均衡。使用SaToken进行统一鉴权，保护 API接口，防止恶意攻击

该系统采用基于**Spring Cloud Alibaba**的微服务架构，前端与后端通过HTTP/RESTful API进行交互。

数据库采用分布式数据库，确保高并发情况下的数据一致性和稳定性。 1.前端客户端：

. 使用 Vue开发web应用，提供用户注册、登录、笔记管理、互动功能等前端页面。

 前端与后端通过 **HTTP API** 进行通信。 2.后端服务端：

后端采用 Spring Boot开发，划分为多个微服务模块：

 **网关服务**：处理所有外部请求，负责路由管理和全局鉴权。

 **认证服务**：负责用户登录、注册与认证，基于 JWT 实现认证流程。  **用户服务**：管理用户信息，提供注册、登录、修改信息等功能。

 **笔记服务**：处理笔记的创建、编辑、删除及权限管理。  **关系服务**：管理用户之间的社交互动，如关注与取关。

 **计数服务**：负责统计用户的社交互动数据，如点赞、粉丝等。

 **OSS 服务**：存储与管理用户上传的图片、视频等非结构化数据。  **消息队列** ：确保异步操作顺序性，减少数据库压力。

 后端服务通过 **Nginx** 进行反向代理，负载均衡。 3.数据库：

 数据库采用 **MySQL** 存储用户信息、笔记信息、互动数据等。  使用 **Redis** 缓存热点数据，如笔记列表、点赞数、收藏数等。

 数据库采用 **ShardingSphere** 进行分库分表，保证数据一致性和高性能。 4.消息队列：

使用**RocketMQ** 实现服务之间的异步通信，如用户互动、统计更新等。 5.对象存储：

笔记中的媒体文件（如图片、视频等）存储在 **MinIO** 等对象存储系统。 6.API网关：

使用 **Spring Cloud Gateway** 实现统一 API 网关，提供流量管理、权限验证、请求路由等功能。

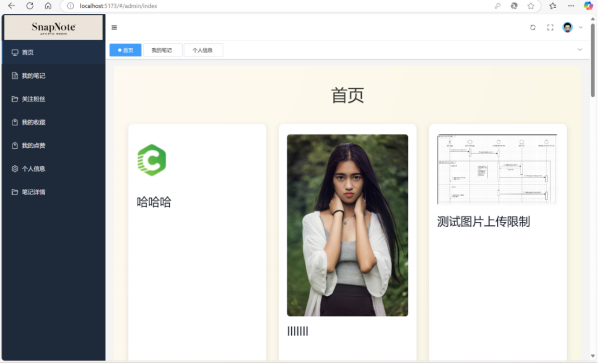
**4.UI界面与交互设计方案**

**4.1登录页面**



以上为我们的登录界面，支持账号登录和密码登录两种登录方式。用户可以根据需要切换，验证码登录 系统会将验证码发到对应邮箱，进行验证码输入60s倒计时。

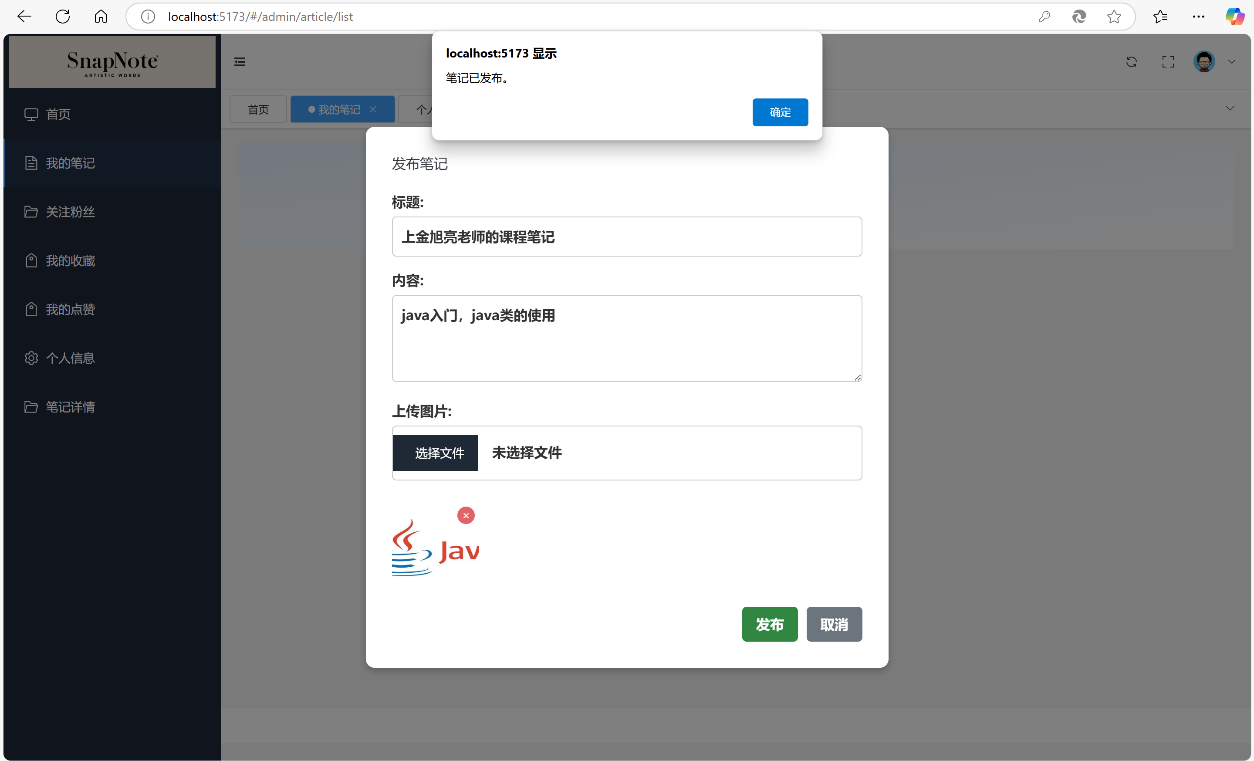
**4.2前端页面展示**



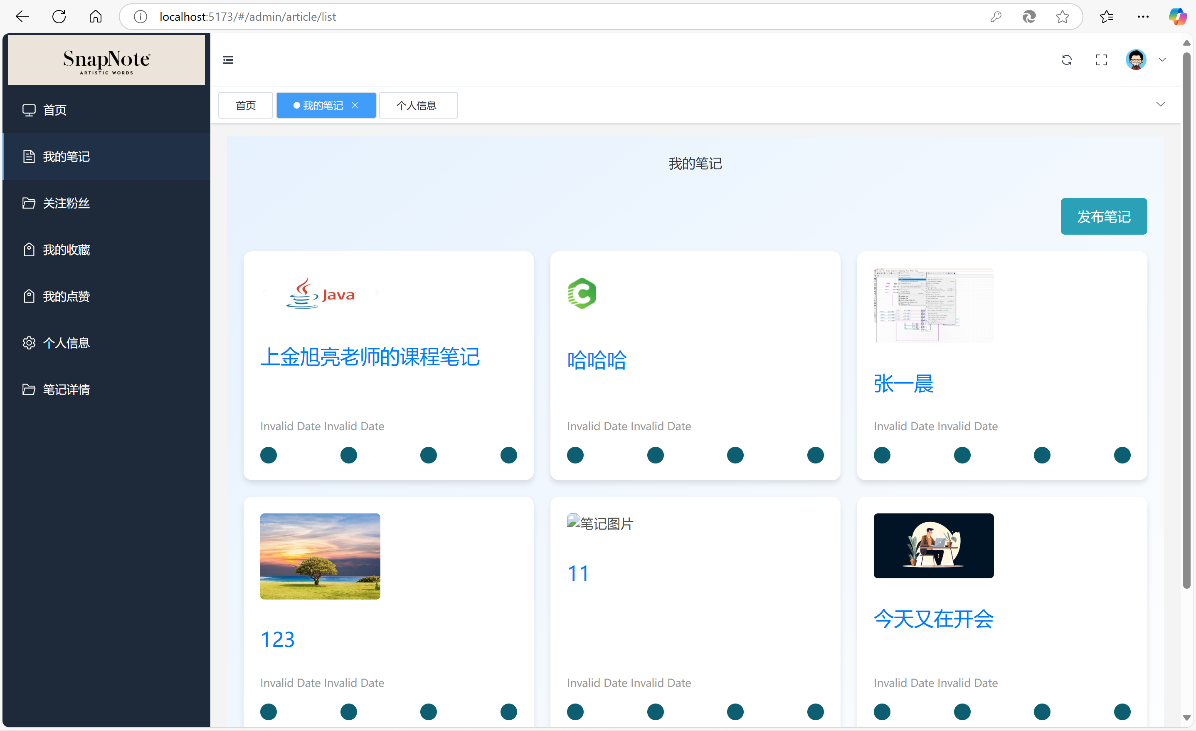
在**首页**可以看到自己发布的笔记与其他用户分享的内容，在上方显示浏览记录，后台可以进行相应推荐 设置。

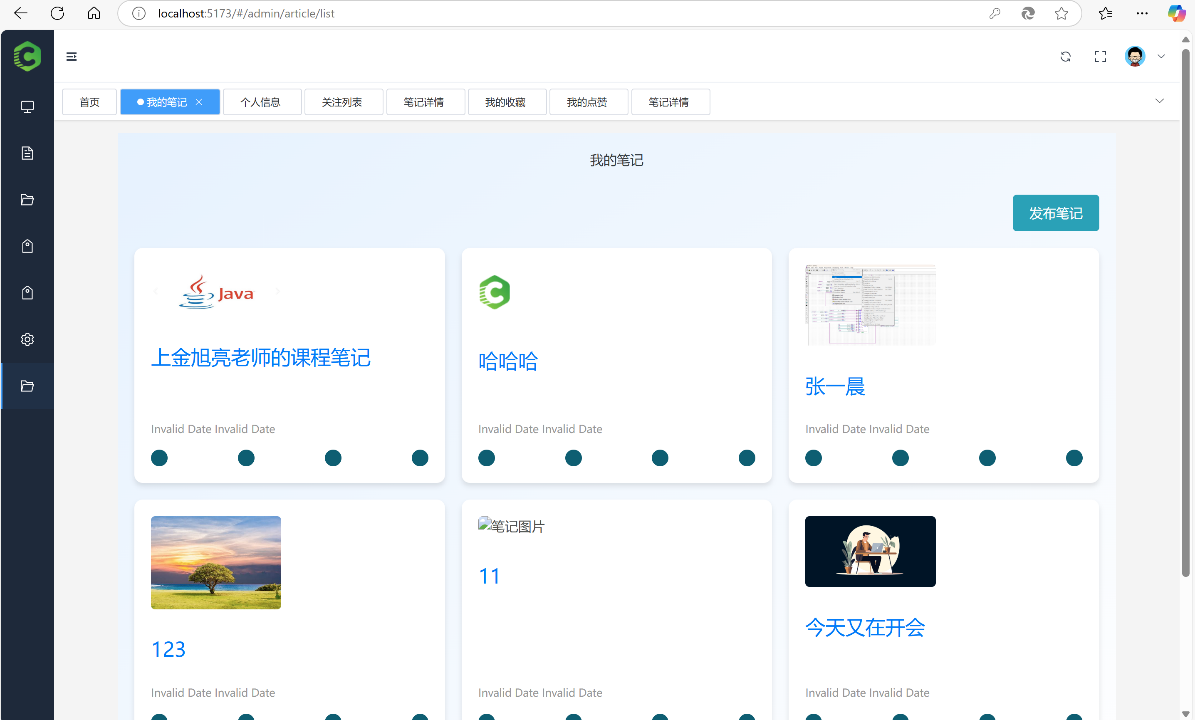
点击侧边栏**我的笔记**，之后点击发布笔记按钮，系统弹出对话框，用户根据需要设置标题，输入内容并 上传相应的图片，点击发布，会显示发布成功消息，笔记出现在我的笔记与首页中。



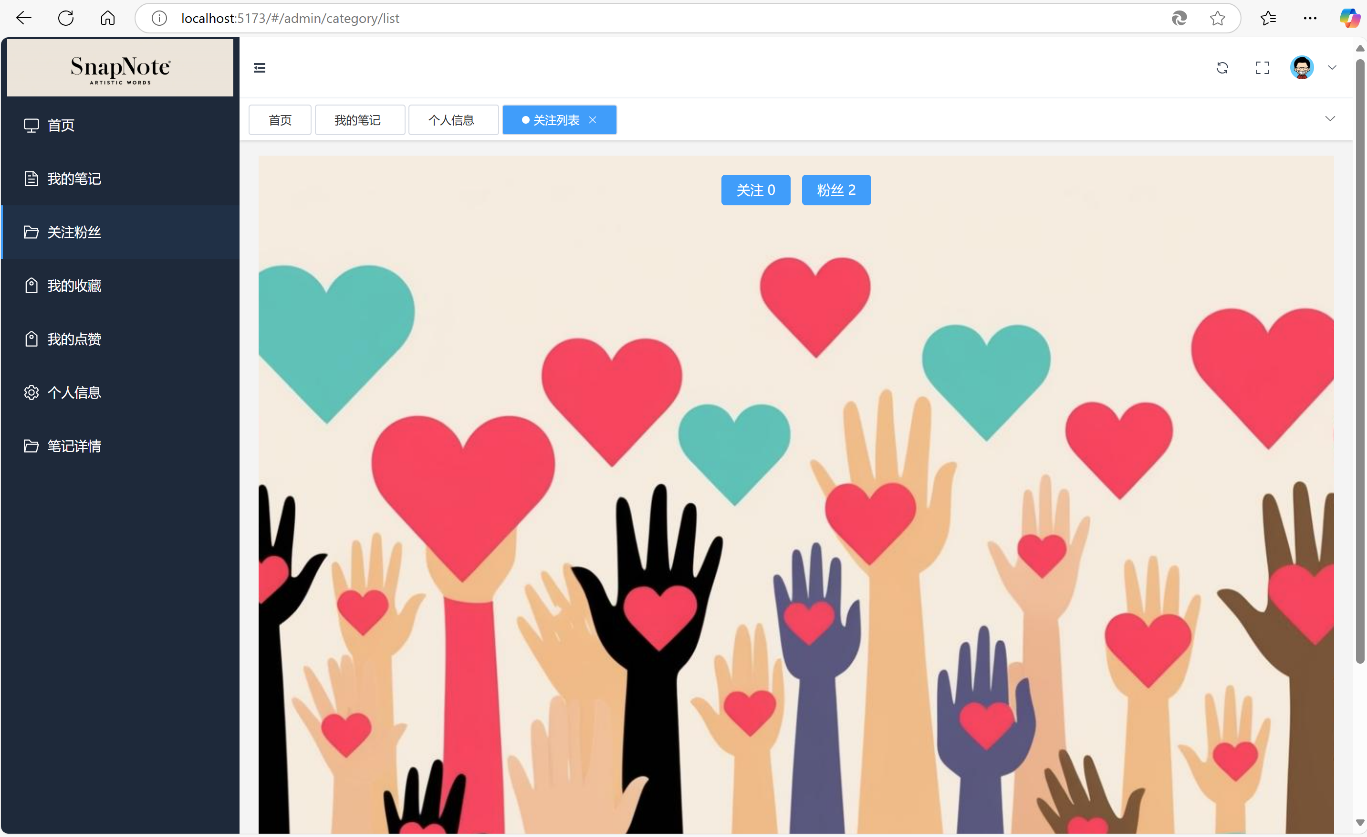


我的笔记和首页里面可以看到笔记已经显示了出来，如下图。

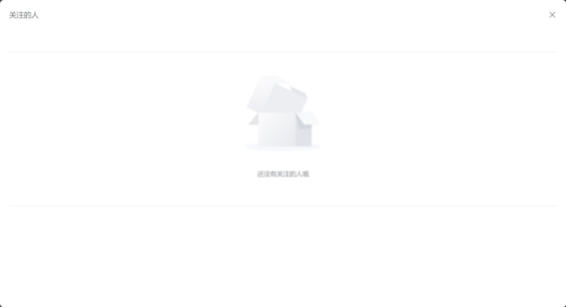




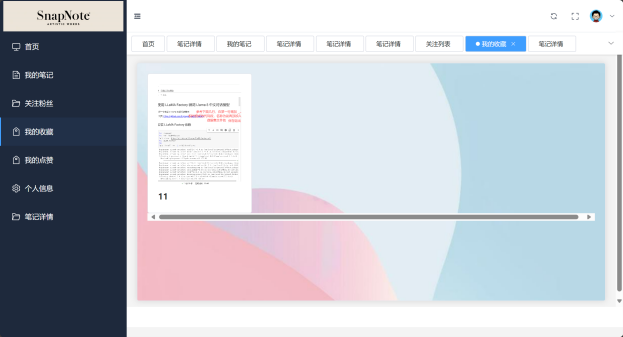
点击侧边栏**关注粉丝**，再次页面可以看到当前用户关注的人数和被关注人数。如下图显示关注0，表示当 前用户并未关注任何人，粉丝2表示有两个粉丝。

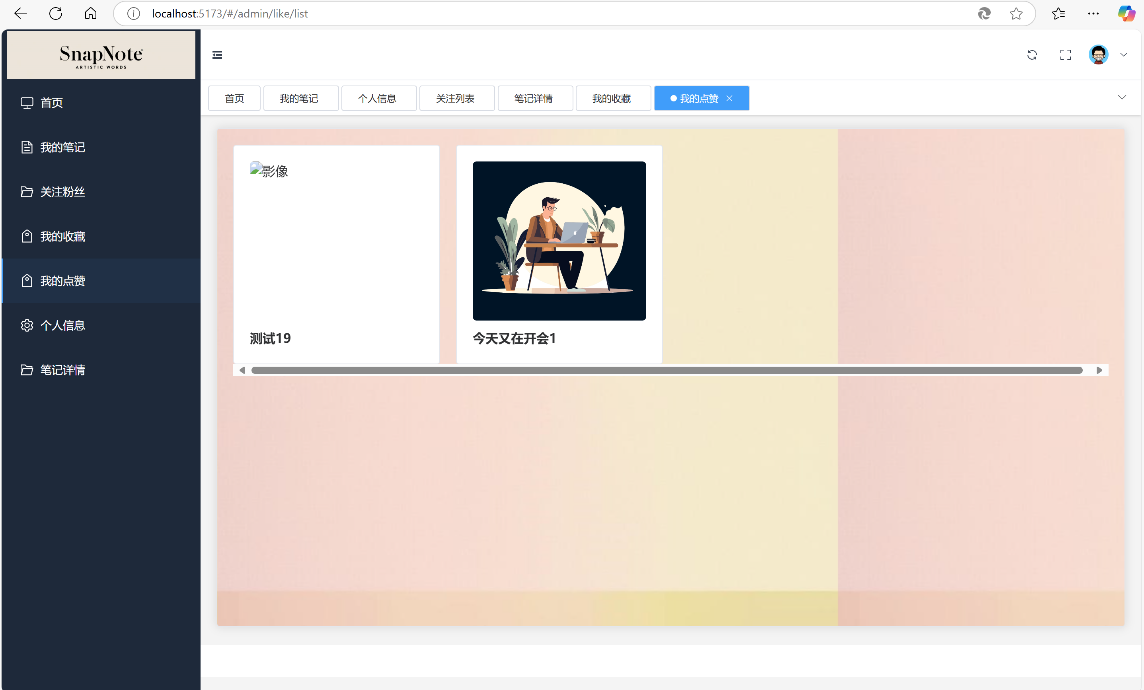


点击关注和粉丝按钮可以查看具体细节，如下图所示。

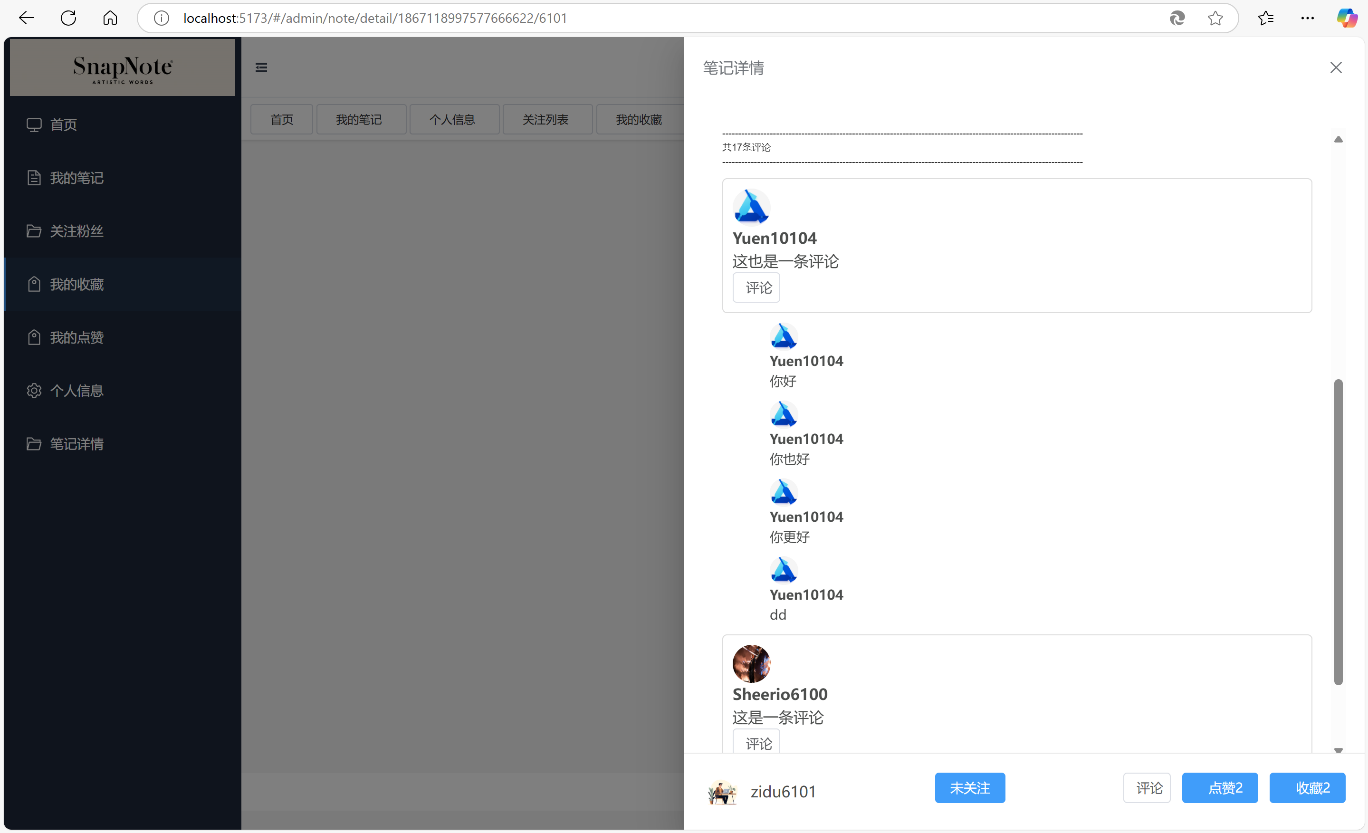


点击侧边栏**我的收藏与我的喜欢**，查看当前用户的收藏与喜欢笔记。

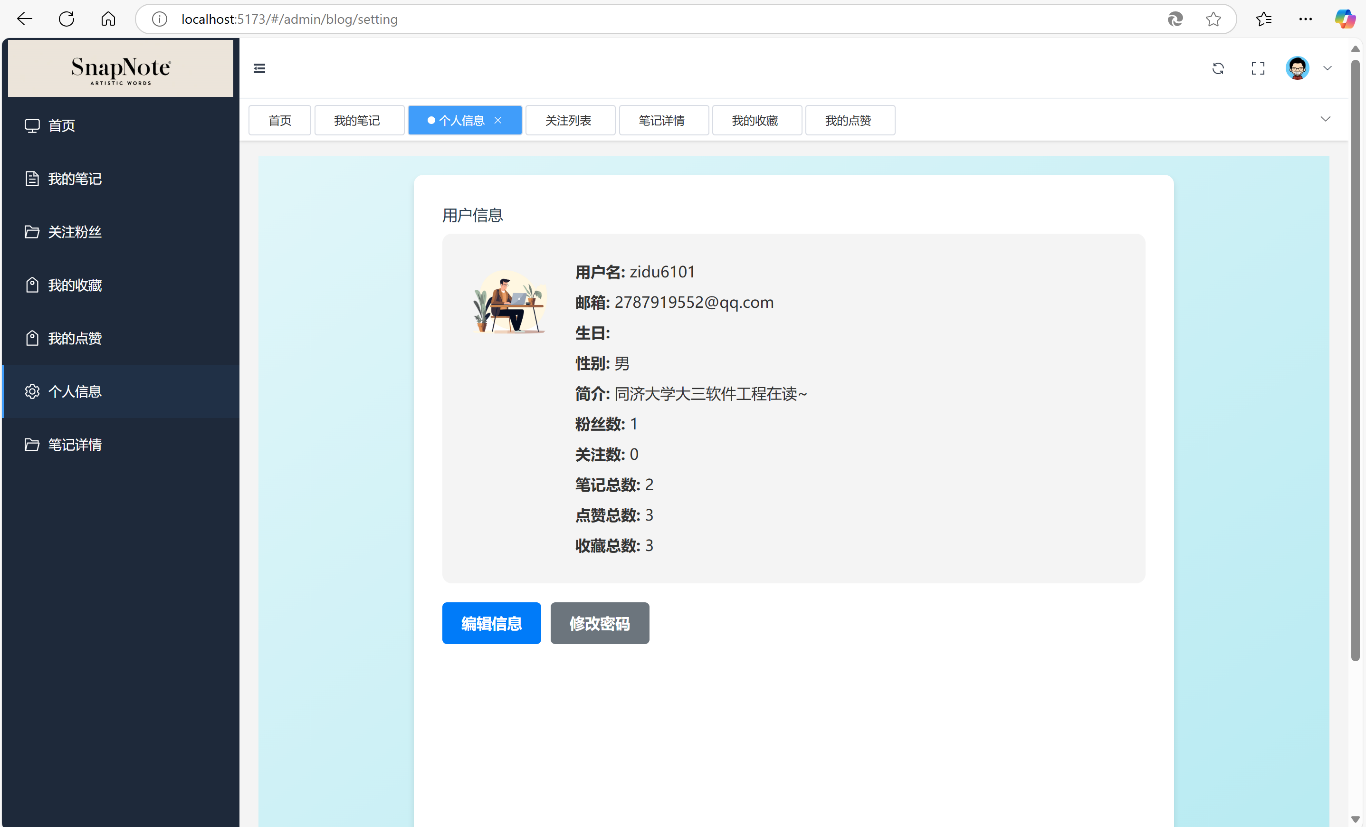




**笔记详情**页面可以通过直接点开首页，我的笔记，我的收藏，我的喜欢等页面显示的笔记跳转，之后会 显示笔记内容和相应的评论，以及评论的评论。

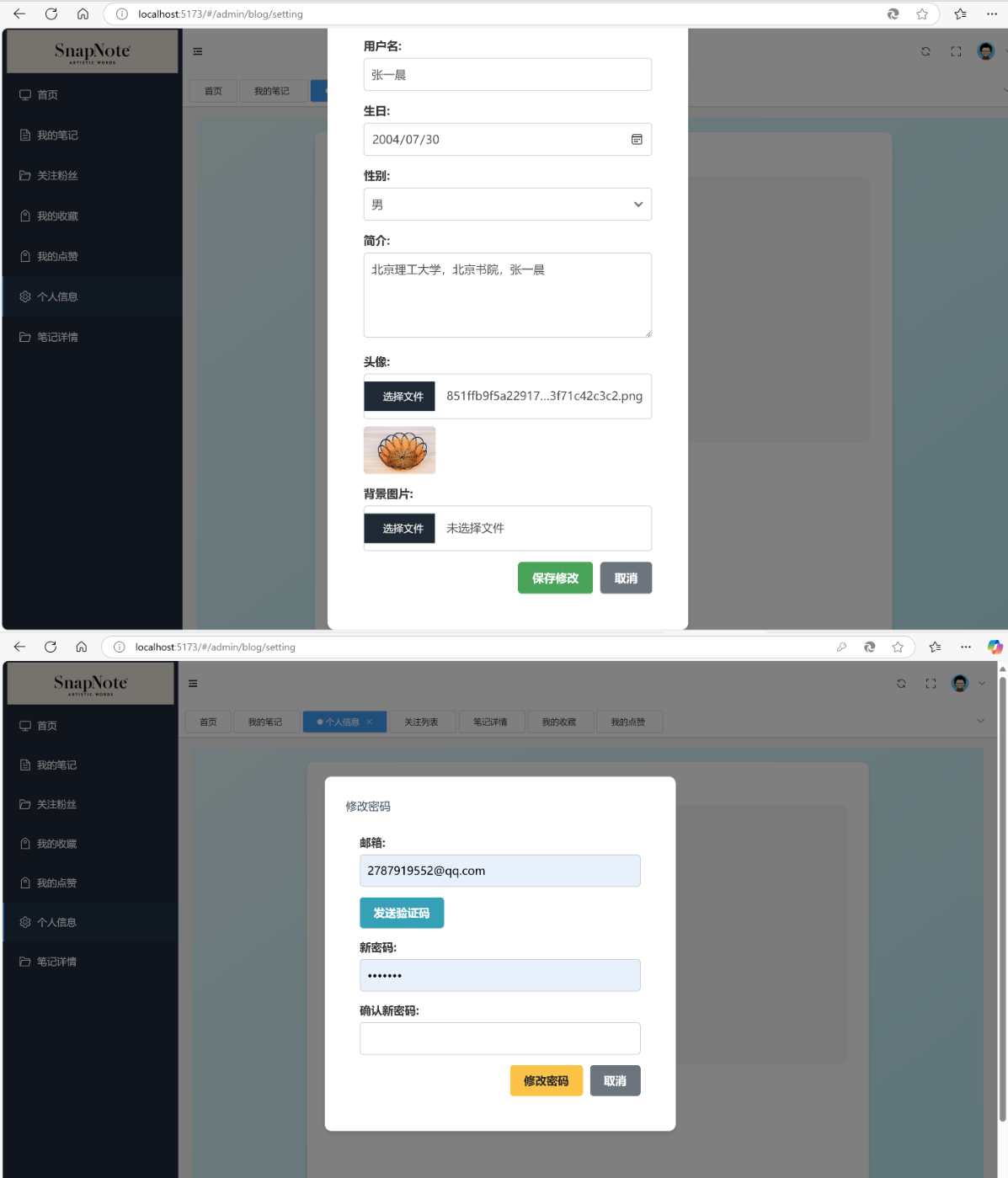


点击侧边栏**个人信息**，用户可以查看和编辑包括用户名、邮箱、生日、性别等基本资料，同时也展示了 个人简介、粉丝数、关注数、笔记总数、点赞总数和收藏总数等社交数据。

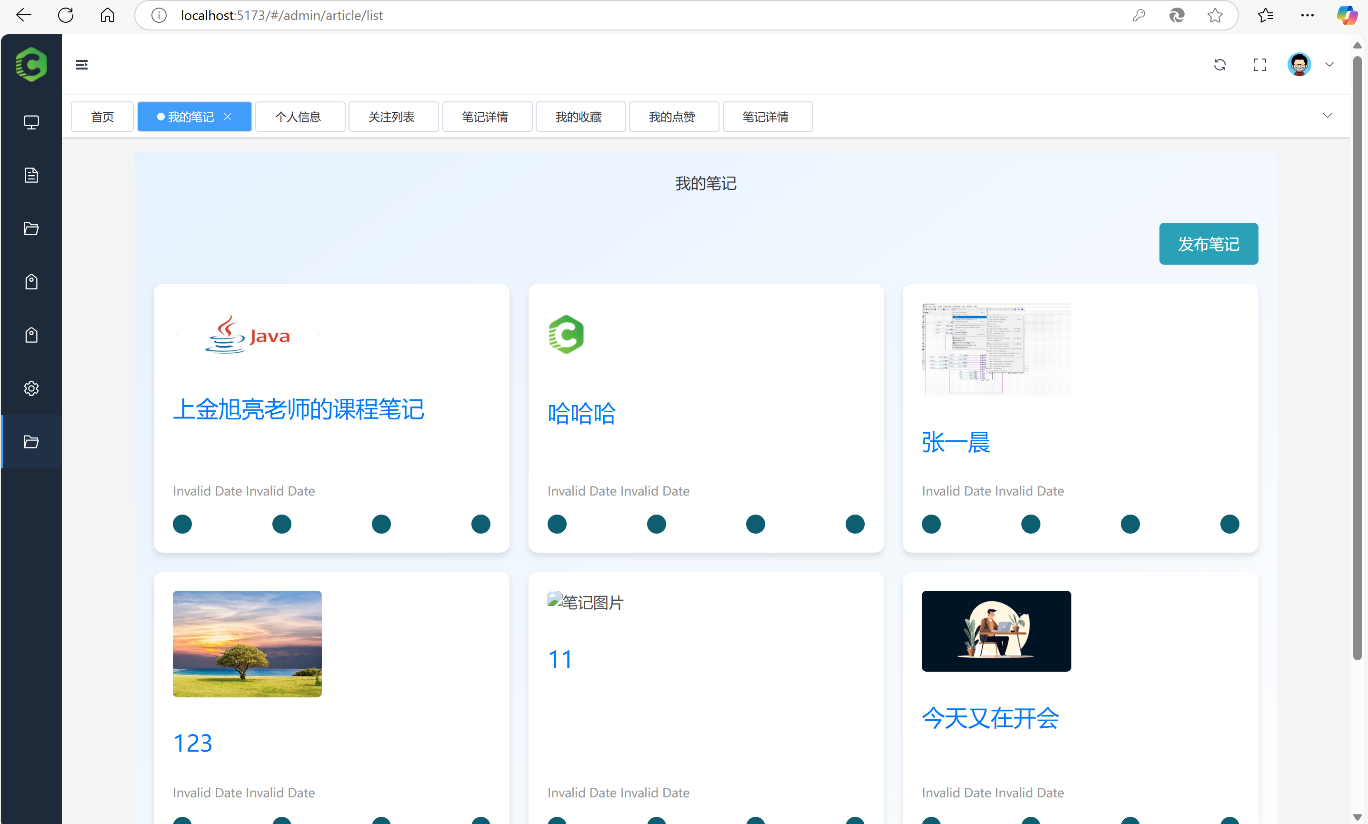


用户可以编辑自己的个人信息，包括用户名，生日，性别，性别，简介，头像，性别等等，修改完毕之 后点击保存修改即可。同样的，用户也可以修改密码，通过邮箱发送验证码方式，修改页面如下图所

示。



同时主页面的侧边栏可以展开和收齐，上面边栏里面的内容可以更方便的定位到某一个路径里面，让我 们更快的打开一个文件。

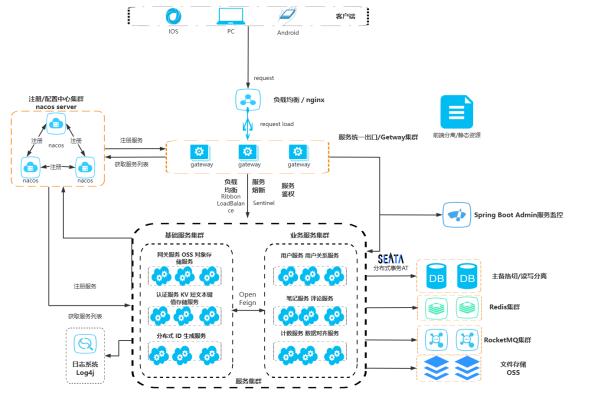


**（三）领域模型**

**1.UML类图展示系统中与业务相关的主要类以及类之间的协作关系（组合、继承、**

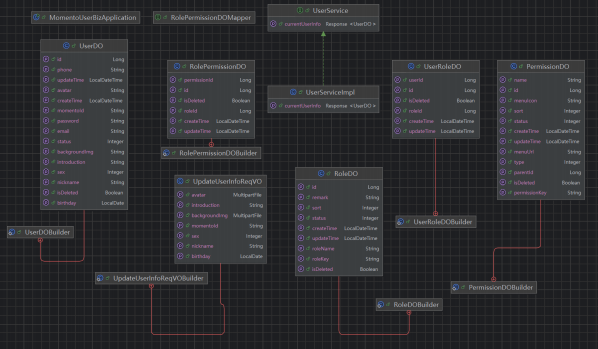
**依赖……）**

**系统体系设计图：**

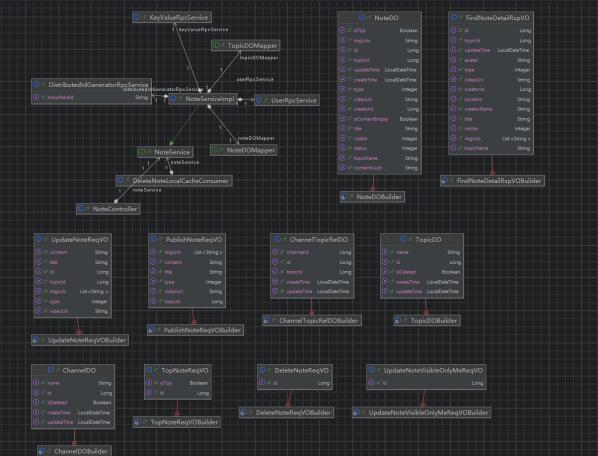


后端采用微服务架构，涉及类较多，因此只给出两个主要服务的相关类图

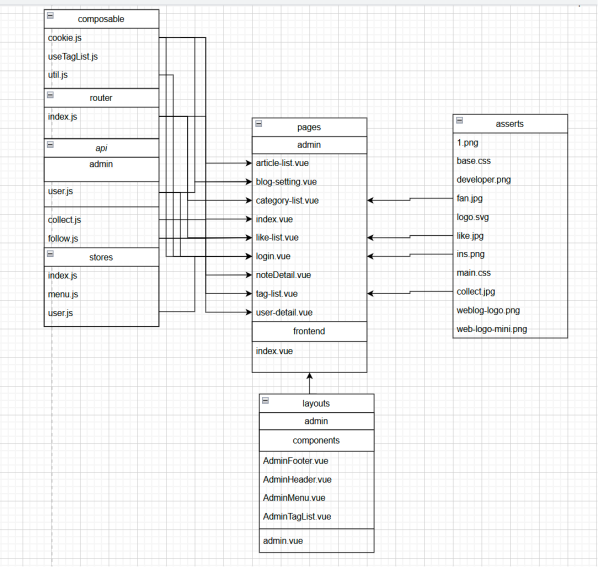
**用户管理模块相关类图：**



**笔记管理模块相关类图**：



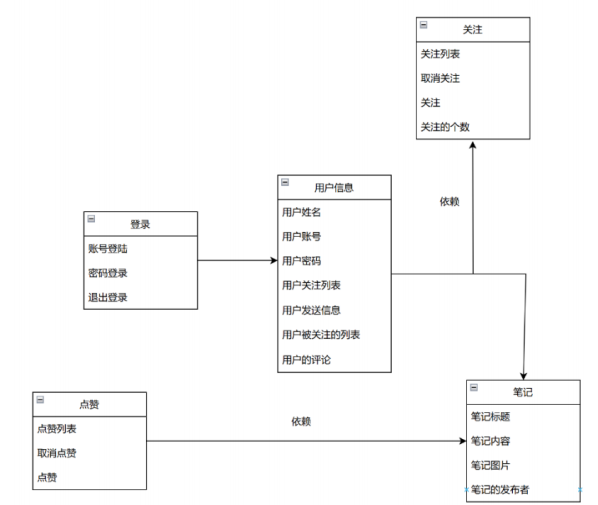
**2.系统总体结构包图**



前端程序包图如上图所示，分为pages ， layouts ，asserts ，composable ， router ，api和stores7个

包，在pages中定义了10个vue文件，对于assert中存储的图片与其他包中的js文件等进行调用。 Api包中 的user.js主要定义有关登录界面的功能， collect.js定义了用户收藏，点赞等功能函数， foolow.js定义了 用户关注与取消关注等功能函数。 Router包中的index.js定义了路由相关信息，包括登录页，首页，后台 首页等。 Stores包中主要包括左边栏菜单的设定，用户退出登录，清除用户信息等功能实现。

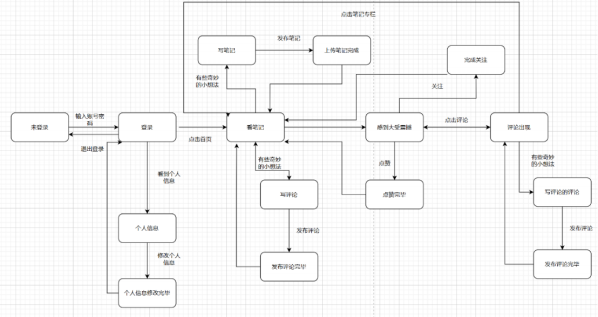
Composable包中cookie.js实现获取token值，删除token，获取Tablist到cookie中等功能，这对于用户 发送笔记到后端很重要，后端端口接收的json文件要包含用户token，否则无法识别用户身份，即发送不 成功。 Utile.js进行确认框的定义，弹出提示内容，用户进行确定取消等。



以上为核心功能结构图，涵盖了用户管理、社交互动、内容发布等多个方面。用户信息模块包括用户的 基本身份数据（如姓名、账号、密码等），并支持登录、退出登录操作。用户可以进行关注操作，包括 关注其他用户、查看自己的关注列表、取消关注、被关注列表、并查看关注的数量。用户可以对内容进 行点赞，还能查看自己的点赞历史，取消点赞。此外，用户还具备评论功能，能够对笔记或其他内容进 行评论。笔记模块则涉及到内容的发布和管理，每篇笔记都包含标题、内容、图片等信息***\*，\****同时也明 确了笔记的发布者，形成了内容创作者与读者的双向互动。

**3.模型关系**

可抽取若干个系统用例，使用活动图、顺序图、状态机图等UML标准图展示系统动态特性，比如数据处 理流程，对象之间的合作关系等。



为了更好的描述我们产品的功能，我们绘制了状态机图。

首先，系统的初始状态是**未登录**，此时用户无法访问个人信息或进行内容发布等操作。用户可以选择输 入账号和密码，进行登录操作。如果用户输入正确的账号和密码，系统会转移到**登录成功**状态，用户此 时可以查看个人信息或进行其他互动操作。若登录失败，则会返回登录界面，继续尝试登录。

用户成功登录之后，他可以选择进入到**个人信息展示**状态，查看或编辑自己的个人资料。此时，用户也 可以选择进入**写笔记**状态，开始撰写新的笔记。用户在**写笔记**状态下完成笔记内容后，点击发布按钮，

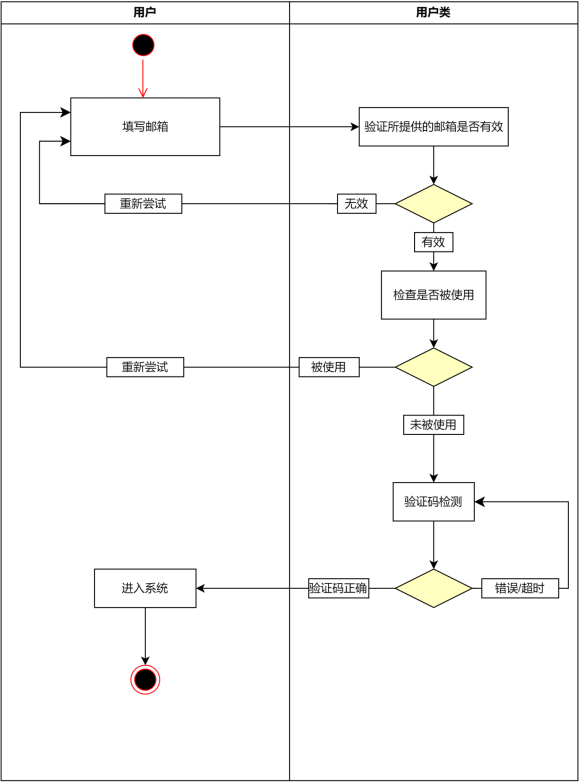
系统会转移到**发布笔记**状态，并上传笔记。上传完成后，用户进入**上传笔记完成**状态，并且回到发布笔 记完成状态。

在用户发布笔记后，他们可以点击进入某篇笔记的**评论区**，查看或互动其他用户的评论。此时用户可以 选择**写评论**，输入评论内容并点击**发布评论**，此时评论被成功保存，系统转移到**评论区**状态，用户可以 继续查看或参与评论互动。如果用户对某个评论有进一步想法，也可以进入**写评论的评论**状态，回复某 个评论，发布评论后，回到**评论区**查看反馈。

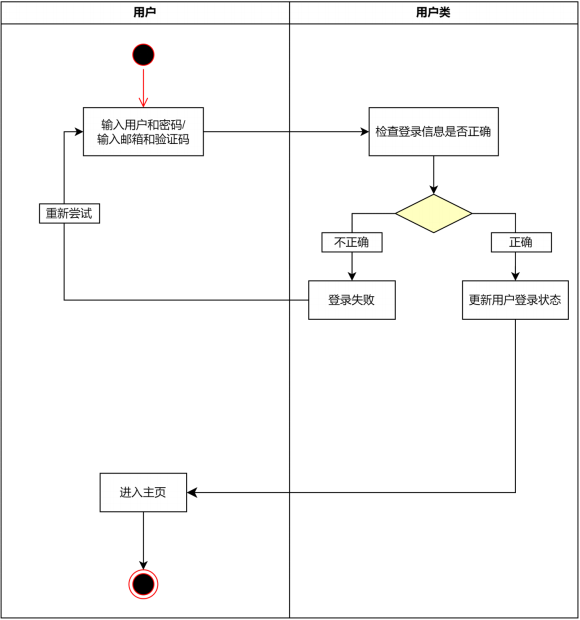
除了评论互动，用户还可以选择**修改个人信息**，进入个人资料编辑界面。在修改完个人资料后，用户进 入**个人信息修改完毕**状态，保存修改并返回到个人信息展示页面。同时，用户也可以选择**退出登录**，退 出当前会话，系统回到**未登录**状态，结束当前的使用流程。

下面给出用例分析：

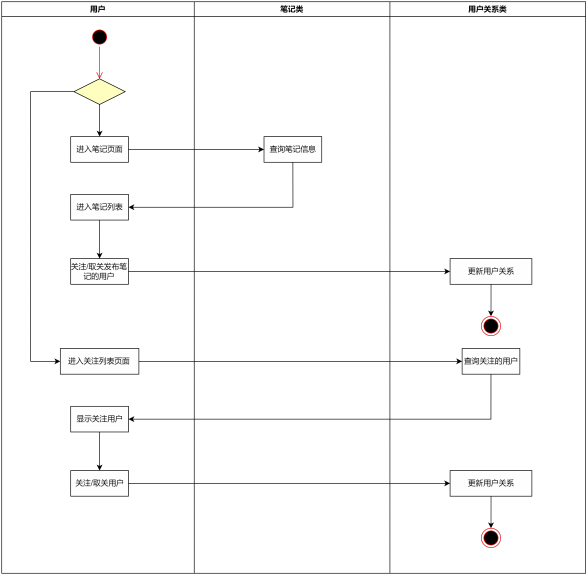
1.用户注册



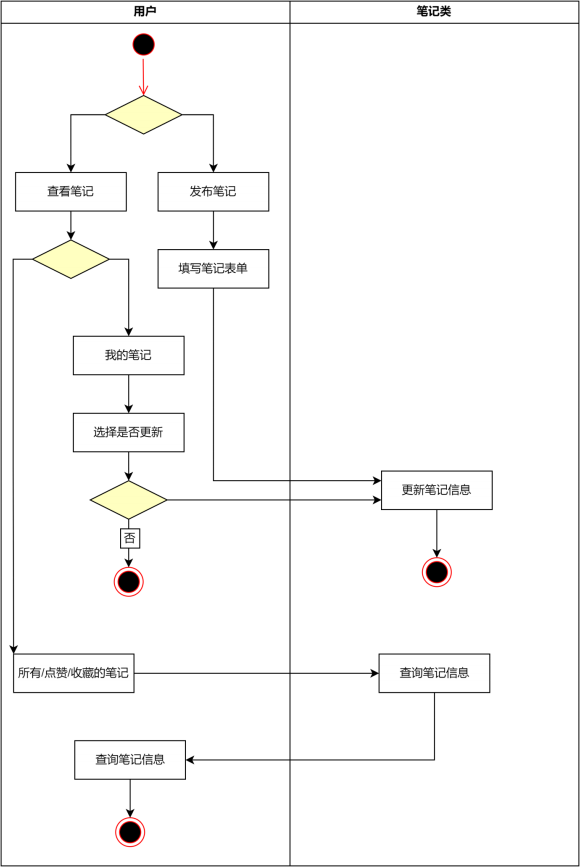
2.用户登录



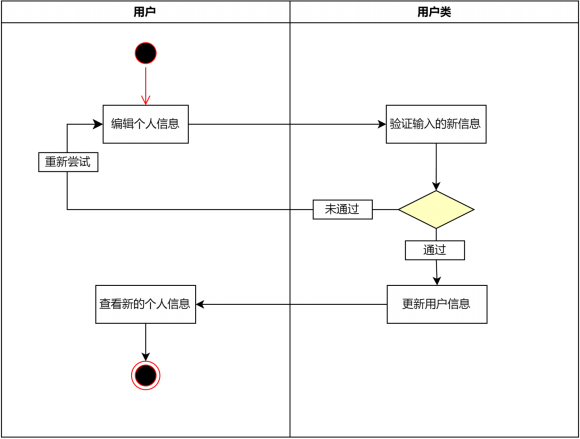
3.关注用户：执行/取消



4.收藏笔记



5.修改个人信息：

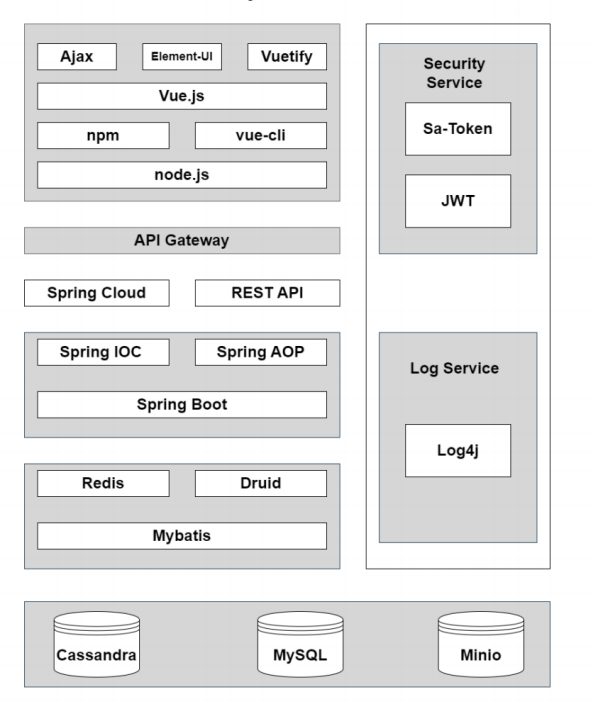


**二、系统设计文档**

**（一）使用组件图、部署图给出系统总体架构概貌。**

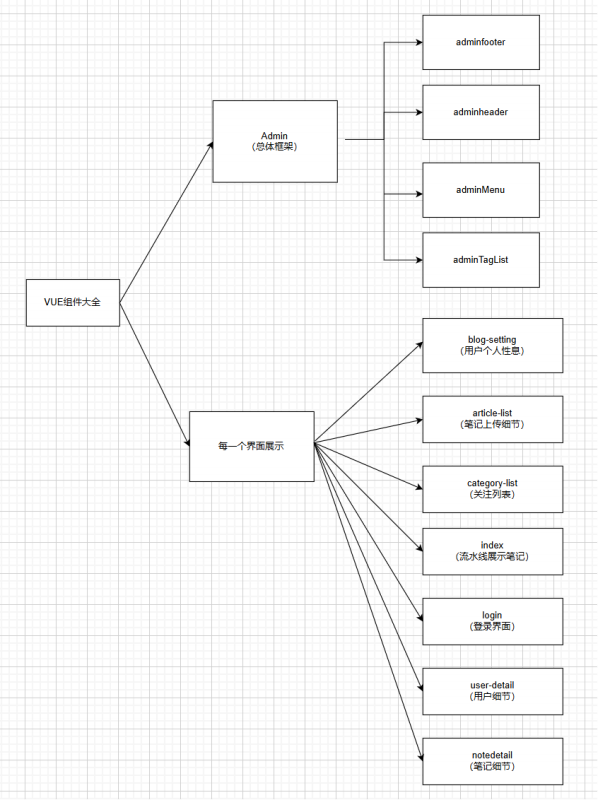
**1.系统将生成多少个组件（比如.jar包，dll，JavaScript模块，Vue/Angular/React组件……），[各组件](af://n82)**[](af://n82)**中主要包容哪些类？**

系统架构大致如下：



**1.web前端**

我们使用的vue组件较多，我们选取了几个比较重要的进行讲解和分析。



article-list.vue包含了用户笔记的增、删、改、查等核心功能。通过 axios 实现与后端的 API 通信，支持 获取用户信息、分页加载笔记、编辑和发布笔记、删除和恢复笔记、图片上传等操作。

blog-setting.vue实现了一个用户信息管理界面，包含用户信息展示、编辑、修改密码等功能。用户信息 展示包括头像、用户名、邮箱、生日、性别、简介以及用户统计数据（如粉丝数、关注数、笔记总数

等)。界面中提供了编辑用户信息的功能，用户可以修改用户名、生日、性别、简介以及上传头像和背景 图片；修改密码功能则通过发送验证码进行验证，确保密码的安全性。

category.vue实现了一个用户关注和粉丝管理功能，用户可以通过抽屉组件查看自己的关注列表和粉丝 列表，并支持无限滚动加载更多数据。每个列表项展示用户的头像、昵称、简介，并提供关注和取消关 注的操作，操作后会同步更新状态。同时，点击用户信息可跳转到该用户的详细页面。

index.vue实现了一个首页笔记展示功能，用户可以查看笔记的缩略图、标题、内容和更新时间。笔记列 表采用瀑布流布局，支持点击查看笔记详情。页面提供了“加载更多”按钮，支持加载更多笔记内容，且显 示加载状态和错误信息。每个笔记的图片、内容等信息在首次加载时被获取，点击笔记后跳转至详情

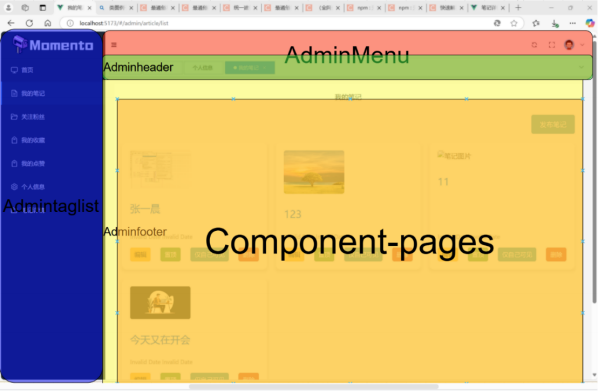
页，同时支持响应式布局，适应不同屏幕大小。

**JavaScript程序如下：**

Axiox.js实例配置，主要用于统一管理请求中的认证 Token，并处理请求和响应的拦截。它在每个请求发 送前自动将获取到的 Token 添加到请求头的 Authorization 字段，并在响应错误时弹出错误提示。通过 这种方式，所有请求都能自动携带 Token ，同时错误处理也被集中管理，简化了请求和错误处理逻辑。

同时我们设置了 Vue Router 的全局前置和后置守卫，前置守卫检查用户是否登录，未登录时禁止访问后 台管理页面并跳转到登录页，已登录时避免重复访问登录页；后置守卫用于动态更新页面标题并隐藏页 面加载动画。

**Vue前端样貌如下：**



以上是不同组件之间的位置关系

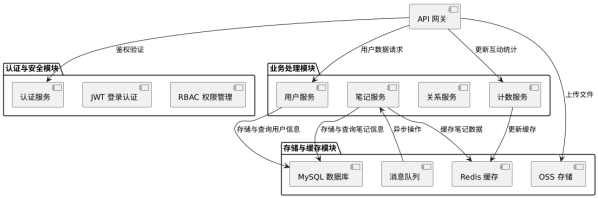
Admin 作为一个综合组件，承载了所有的admin开头的子组件。 AdminMenu 位于页面的顶部，除了显 示登录按钮，还负责展示其他导航菜单，并且还展示登录信息和退出登录的界面； AdminHeader 是

AdminMenu 的子组件，位于其下方，用于显示已经打开的页面的标题和一些辅助信息； AdminTagList 则在页面的左侧，存放一系列用于快速跳转或分类选择的按钮； AdminFooter 位于页面底部，主要用于 设置背景。中间一部分是其他组件的位置，用于放置用户打开的组件界面。

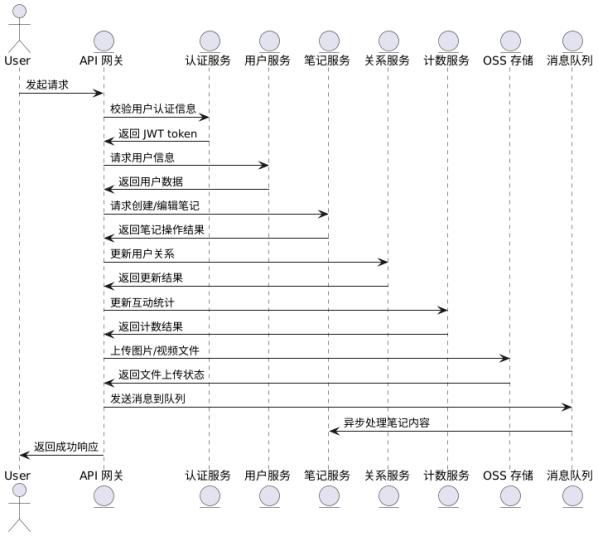
**2.web后端**

**1.系统组件图：**

系统中各个主要模块及其之间的关系，帮助理解每个服务如何协同工作，满足系统的高可用性与可扩展 性需求。例如，用户服务、笔记服务、关系服务等通过 API 网关对外提供服务，同时使用消息队列进行 异步任务处理，确保系统在高并发环境下的流畅运行。



各个微服务的关系如下图所示：



**2.以下是后端核心服务：**

 **网关服务**：处理所有外部请求，负责路由管理和全局鉴权。

 **认证服务**：负责用户登录、注册与认证，基于 JWT 实现认证流程。

 **用户服务**：管理用户信息，提供注册、登录、修改信息等功能。  **笔记服务**：处理笔记的创建、编辑、删除及权限管理。

 **关系服务**：管理用户之间的社交互动，如关注与取关。

 **计数服务**：负责统计用户的社交互动数据，如点赞、粉丝等。

 **OSS 服务**：存储与管理用户上传的图片、视频等非结构化数据。  **消息队列** ：确保异步操作顺序性，减少数据库压力。

每个微服务会打包为一个jar包

**3.服务拆分**

笔记管理、用户管理、认证、用户关系管理等被设计为独立服务，支持独立开发和分布式部署。

**4.分层架构**

通过逻辑分离将系统划分为不同的层次，每一层通过明确定义的接口进行通信 系统层次：

 **表现层**：用户通过基于Vue的前端与系统交互，通过HTTP请求将数据发送到后端网关。 Nginx处理 反向代理和负载均衡。

 **网关层** ：**Spring Cloud Gateway**作为统一入口，处理认证、流量控制和动态路由。  **业务逻辑层**：每个功能模块作为独立的微服务运行，处理核心业务逻辑。

 **数据访问层**：集成多种存储系统（MySQL、 Redis、Cassandra、 MinIO）来优化不同数据类型和访 问模式的存储。

**5.各个微服务的详细解释：**

**a. Authentication Service（认证服务）**

 **技术栈** ：**Spring Boot**, **Spring Security**, **SaToken**  功能：

。 用户注册与登录（支持密码登录和验证码登录） 。 用户认证（生成和校验JWT Token）

。 密码重置

 **存储**：用户认证信息存储在**MySQL**中，登录状态缓存于**Redis**中。

**b. User Service（用户服务）**

 **技术栈** ：**Spring Boot**, **MyBatis**, **Redis**  功能：

。 用户信息管理（查询、修改个人信息、更新密码） 。 用户关系管理（关注与取关）

 **存储**：用户信息存储在**MySQL**中，用户关系和部分高频访问数据缓存于**Redis**中。

**c. Note Service（笔记服务）**

 **技术栈** ：**Spring Boot**, **MyBatis**, **Redis**, **Cassandra**  功能：

。 笔记发布、编辑、删除

。 笔记浏览与查询（支持标签、关键词过滤） 。 笔记置顶与权限设置

 **存储**：笔记索引存储在**MySQL**中，笔记内容存储在**Cassandra**中，媒体文件存储在**Minio**对象存储 中，热点数据缓存于**Redis**中。

**d. Interaction Service（互动服务）**

 **技术栈** ：**Spring Boot**, **Redis**, **RocketMQ**  功能：

。 点赞、收藏操作的处理（高并发写入） 。 关注与取关操作的处理

。 用户互动数据的缓存与异步落库

 **存储**：互动数据主要存储在**Redis**中，通过**RocketMQ**实现异步落库至**MySQL**。

**e. Count Service（计数服务）**

 **技术栈** ：**Spring Boot**, **Redis**, **MySQL**  功能：

。 统计用户的关注数、粉丝数、点赞数、收藏数等 。 统计笔记的点赞数、评论数、收藏数等

。 提供实时统计数据接口

 **存储**：统计数据存储在**Redis**和**MySQL**中，确保高并发下的高性能。

**2.这些组件如何部署（哪些放在Server端，哪些放在Client端，使用几台服务器， 每台服务器上部署哪些组件……）？**

**将单个微服务进行打包，然后将jar上传**

在部署方面，系统采用**Nginx**作为主要的负载均衡器，通过轮询、最少连接和IP哈希等策略，将客户端请 求高效分配到后端多台服务器，确保系统的可用性和扩展性。为了提高负载均衡器的高可用性，部署了 多个Nginx实例，并结合**Keepalived**实现主备切换，避免单点故障。

系统架构采用**微服务架构**，包含基础功能服务器、中间件服务器和业务逻辑服务器等多个独立服务。所 有微服务通过**Nacos**进行注册和管理，支持动态服务发现和健康检查，确保只有健康的服务实例处理请 求。同时， Nacos集中管理各微服务的配置，支持动态刷新和灰度发布，减少配置变更带来的停机时间。 外部请求统一通过**API网关**（如Zuul或Spring Cloud Gateway）进行路由、鉴权和限流，内部服务间通 信则采用HTTP/REST或gRPC协议，确保高效低延迟。

**数据库与缓存**方面，主要使用**MySQL**和**Cassandra**。 MySQL作为关系型数据库，采用主从架构实现读写 分离，支持复杂事务和查询操作； Cassandra作为分布式NoSQL数据库，适用于大规模数据和高并发读 写，提供高可用性和横向扩展能力。缓存系统方面，使用**Redis**进行热点数据缓存，采用主从复制架构实 现读写分离，并通过合理的缓存策略（如LRU、TTL）提高缓存命中率，减少数据库访问。

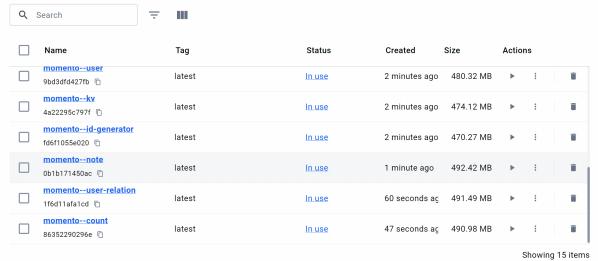
**搜索与消息队列**部分，采用**Elasticsearch**作为分布式搜索引擎，支持高效的全文搜索和实时数据分析， 通过日志收集和数据同步机制将业务数据实时同步至Elasticsearch，确保搜索的实时性和准确性。消息 队列选用**RocketMQ**，适用于订单处理、通知发送和日志收集等异步处理场景，配置了消息持久化和重 试机制，确保消息的可靠传输和处理。

文件存储方面，采用**分布式文件系统**（如HDFS或Ceph）或**对象存储服务**（如MinIO、阿里云OSS、AWS S3）进行文件数据的存储，确保文件的安全性和高可用性。定期备份文件存储库，并制定数据恢复策

略，确保在数据丢失或损坏时能够快速恢复。

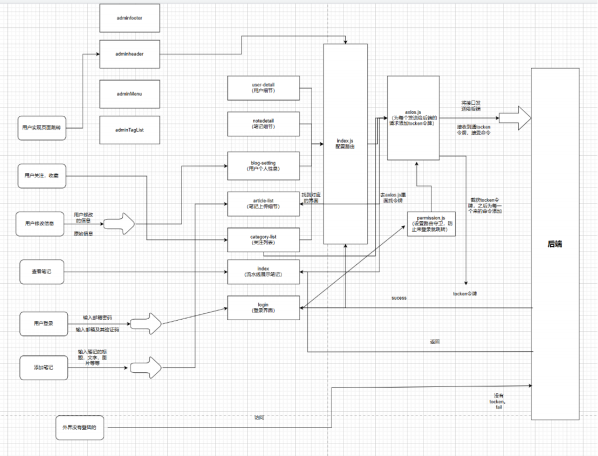
**利用Docker部署各个微服务**

可以在docker中查看已经搭建好的微服务：



**（二）本系统中的技术关键点及解决方案，可通过类图、顺序图等介 绍**

**1.登录前端设计**



具体步骤如下所示

**1.1用户登录界面**

用户访问系统时，首先会看到登录界面。在这个界面上，用户需要输入自己的账号和密码。如果是第一 次登录，系统会提示自动注册。如果用户选择验证码登录，则会使用邮箱验证码进行验证。此界面设计 简洁，包含了邮箱输入框、密码输入框、验证码输入框和登录按钮，同时提供了登录方式的切换功能。

**1.2用户输入账号密码， login.vue 发送给后端**

用户在登录界面输入自己的账号（通常为邮箱）和密码后，前端会将这些信息通过表单提交给后端。此 时，前端会根据用户选择的登录方式（密码登录或验证码登录）构造请求。对于密码登录，前端会发送 包含邮箱和密码的请求；对于验证码登录，则发送邮箱和验证码信息。请求发送至后端的登录接口，后 端将进行验证。

**1.3后端响应，如果账号密码正确，返回一个 Token 令牌，交给 axios.js 保存**

后端收到请求后，会根据传递的账号和密码进行验证。如果验证成功，后端会生成一个 JWT（Json Web Token）令牌作为用户身份的凭证，并将该令牌作为响应返回给前端。前端会使用 axios.js 对返回的

Token 进行处理，存储在合适的地方，并确保后续的 API 请求都会自动携带这个令牌进行身份验证。

**1.4index.js 实现页面的跳转，使其到达主界面**

在成功登录后，前端会根据返回的 Token 将用户信息存储在全局状态中，并跳转到系统的主界面（通常 是后台管理系统的首页）。这个跳转操作通过 Vue Router 实现，确保登录成功后，用户能够顺利进入到

系统的主页面。同时，系统会确保每次请求都携带 Token ，以便后端能够验证用户的身份。

**2.其他基本功能的前端设计**

**2.1用户修改笔记**

用户在系统中修改笔记时，首先输入新的笔记标题和其他相关信息。前端通过 axios.js 获取当前用户的 Token 令牌，并将修改后的内容（如笔记标题、内容等） 一起发送给后端进行处理。后端接收到请求

后，会验证用户的身份信息（通过 Token 令牌），并根据修改的内容更新数据库中的笔记数据。如果操 作成功，后端返回相应的成功信息，前端根据该信息更新界面，并给用户反馈成功或失败的提示。

**2.2用户点赞、关注操作**

当用户进行点赞、关注等互动操作时，系统会首先获取当前用户的 Token 令牌，然后将点赞或关注的相 关信息（如目标笔记、文章或用户的 ID）发送给后端。操作完成后，后端会返回相应的操作结果，前端 根据返回的结果更新界面，展示新的点赞、关注列表。

**2.3用户界面跳转**

在用户登录、修改信息或完成其他操作后，系统需要根据操作结果实现页面跳转。前端通过 index.js 配 置的 Vue Router 来实现跳转功能。

**3.数据库设计**

**1. 用户表（user）**

|  |  |  |
| --- | --- | --- |
| **字段名** | **数据类型** | **描述** |
| id | INT | 用户ID，主键 |
| username | VARCHAR(255) | 用户名 |
| password | VARCHAR(255) | 密码 |
| email | VARCHAR(255) | 邮箱 |
| phone | VARCHAR(20) | 手机号码 |
| avatar | VARCHAR(255) | 头像链接 |
| created\_at | DATETIME | 用户注册时间 |
| updated\_at | DATETIME | 用户信息更新时间 |

**2. 笔记表（note）**

|  |  |  |
| --- | --- | --- |
| **字段名** | **数据类型** | **描述** |
| id | INT | 笔记ID，主键 |
| user\_id | INT | 用户ID（外键） |
| content | TEXT | 笔记内容 |
| media | VARCHAR(255) | 媒体链接 |
| tags | VARCHAR(255) | 标签 |
| created\_at | DATETIME | 笔记创建时间 |
| updated\_at | DATETIME | 笔记更新时间 |

**3. 点赞表（like）**

|  |  |  |
| --- | --- | --- |
| **字段名** | **数据类型** | **描述** |
| id | INT | 点赞ID，主键 |
| user\_id | INT | 用户ID（外键） |
| note\_id | INT | 笔记ID（外键） |
| created\_at | DATETIME | 点赞时间 |

**4. 收藏表（collect）**

|  |  |  |
| --- | --- | --- |
| **字段名** | **数据类型** | **描述** |
| id | INT | 收藏ID，主键 |
| user\_id | INT | 用户ID（外键） |
| note\_id | INT | 笔记ID（外键） |
| created\_at | DATETIME | 收藏时间 |

**5. 关注表（follow）**

|  |  |  |
| --- | --- | --- |
| **字段名** | **数据类型** | **描述** |
| id | INT | 关注ID，主键 |
| follower\_id | INT | 关注者ID（外键） |
| followed\_id | INT | 被关注者ID（外键） |
| created\_at | DATETIME | 关注时间 |

**6. 用户计数表（user\_count）**

|  |  |  |
| --- | --- | --- |
| **字段名** | **数据类型** | **描述** |
| id | INT | 主键，自增 |
| user\_id | INT | 用户ID（外键，关联用户表） |
| following | INT | 关注数 |
| followers | INT | 粉丝数 |
| likes | INT | 点赞数 |
| favorites | INT | 收藏数 |
| created\_at | DATETIME | 数据创建时间 |
| updated\_at | DATETIME | 数据更新时间 |



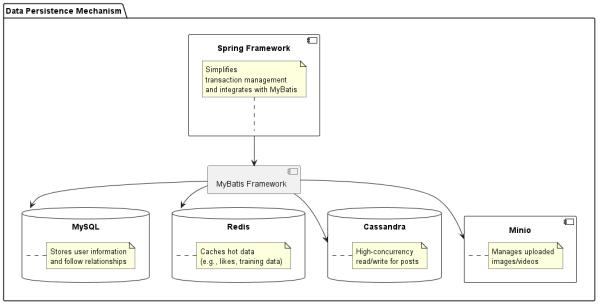
**7. 笔记计数表（note\_count）**

|  |  |  |
| --- | --- | --- |
| **字段名** | **数据类型** | **描述** |
| id | INT | 主键，自增 |
| note\_id | INT | 笔记ID（外键，关联笔记表） |
| likes | INT | 点赞数 |
| favorites | INT | 收藏数 |
| comments | INT | 评论数 |
| views | INT | 浏览数 |
| created\_at | DATETIME | 数据创建时间 |
| updated\_at | DATETIME | 数据更新时间 |

**8. 评论表（comment）**

|  |  |  |
| --- | --- | --- |
| **字段名** | **数据类型** | **描述** |
| id | INT | 主键，自增 |
| note\_id | INT | 笔记ID（外键，关联笔记表） |
| user\_id | INT | 用户ID（外键，关联用户表） |
| content | TEXT | 评论内容 |
| parent\_id | INT | 父评论ID（用于回复的评论） |
| created\_at | DATETIME | 评论创建时间 |
| updated\_at | DATETIME | 评论更新时间 |
| deleted\_at | DATETIME | 删除时间（软删除） |

下面是数据库与存储的示例图



**4.后端接口设计**

API的管理通过apifox进行

**4.1 统一响应格式**

所有接口返回数据均采用统一响应格式，确保前端可以统一解析，简化错误处理和数据管理。 **响应结构**：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **名称** | **类型** | **必选** | **约束** | **中文名** | **说明** |
| success | boolean | true | none | 响应状态 | true表示成功， false表示失败 |
| message | string | true | none | 响应消息 | 错误时返回的错误消息 |
| errorCode | string | true | none | 错误码 | 错误时返回的异常码 |
| data | object | true | none | 响应数据 | 请求成功时返回具体的数据 |

**4.2 统一请求头**

请求头需包含认证信息，所有需要身份验证的接口都应在请求头中传递 Authorization 字段。该字段 的值应为登录成功后获取的 JWT token 。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **名称** | **位置** | **类型** | **必选** | **说明** |
| Authorization | header | string | 是 | 必须提供的 JWT token ，用于身份验证 |

**4.3 接口命名规范**

接口命名应简洁明了，采用RESTful风格，是名词复数形式，路径采用小写字母，多个单词使用连字符 （ - ）分隔。

 **资源的命名** ：使用名词的复数形式，例如：  /users ：获取用户列表

 /users/{userId} ：获取指定用户信息

 /posts ：获取所有笔记

 /posts/{postId} ：获取指定笔记  **接口路径设计**：

 获取资源： GET /resource

 创建资源： POST /resource

 更新资源： PUT /resource/{id}

 删除资源： DELETE /resource/{id}

 搜索/过滤资源： GET /resource?filter=value

 批量操作： POST /resource/batch

 认证相关操作： POST /auth/login ， POST /auth/logout

**4.4 分页规范**

对于返回多个资源的接口，支持分页查询，避免一次性返回大量数据。  分页参数：

 page ：当前页码，默认为1

 size ：每页显示的资源数量，默认为10

 sort ：排序字段，例如： createdAt ， -createdAt （降序）

**5.微服务代码架构**

微服务代码大致架构如下

|  |
| --- |
| 1 config  2 controller  3 domain  4 enums  5 exception  6 service |

**1.配置相关， config**

每个服务通常需要不同的配置（如数据库连接、消息队列、缓存、 API 网关地址等），将这些配置集中到

config 包中，便于维护和管理。

**2.控制器层controller**

Controller 是系统的**入口点**，它负责从客户端接收 HTTP 请求（如 GET 、 POST 、 PUT 、 DELETE 等），并根据请求的路径、参数和方法调用对应的业务逻辑。

控制器可以从客户端请求中提取各种类型的参数，例如：

 URL 路径参数

. 查询参数（Query Parameters） . 请求体（Request Body）

 表单参数

控制器的主要职责是将客户端请求传递给 **服务层（Service Layer）** ，然后将服务层的返回结果返回给客 户端。 **Controller 不应该包含具体的业务逻辑**，这有助于代码的分层和职责的明确。

**3.领域模块（Domain Layer）**

domain 层专注于系统的业务逻辑和数据模型，它是架构中连接 **数据层**（持久化层） 和 **应用层**（服务 层）的桥梁。它包含与业务密切相关的代码，例如：

 数据对象（实体）

. 数据访问对象（DAO/Mapper）  业务规则和领域逻辑

domain **层与其他层的关系**

**(1) 与** Controller **层**

 Controller 层接收到请求后，会通过 Service 层调用 domain 层，获取需要的数据。  domain 层专注于业务数据和规则，而不直接处理 HTTP 请求。

**(2) 与** Service **层**

 Service 层是业务逻辑的核心，但它的实现通常依赖于 domain 层的领域模型和数据访问。  Service 调用 domain 层的组件（如 Mapper 或 DO），完成业务流程。

**(3) 与数据库**

 domain 层通过 Mapper 或 DAO 实现与数据库的直接交互，是持久化操作的实现部分。

**（4）模型层**

1. dto （数据传输对象） :

 示例： CollectUnCollectNoteMqDTO 、 LikeUnlikeNoteMqDTO :

 包含用于消息队列传输的数据结构。

 作用：用于封装生产者和消费者之间传递的消息内容。

2. vo （视图对象） :

 示例： CollectedNoteVO 、 CommentRspVO :  封装接口返回给前端的数据结构。

 作用：用于定义接口的响应数据结构，统一返回格式。  示例： CommentReqVO 、 DeleteNoteReqVO :

 封装接口请求的参数。

 作用：用于定义前端请求的参数结构，统一处理请求数据。

**（5）** enums **:**

 定义系统中使用的枚举类型。

 作用：用于定义固定的枚举值，例如状态码、操作类型等。

（ 6） exception **:**

 包含自定义异常类。

 作用：处理系统中可能出现的业务或系统异常，提供更友好的错误提示。

**6.缓存优化**

 **热点数据缓存**：将高频访问的数据（如用户信息、笔记列表、点赞数、收藏数等）缓存于**Redis** 中，减少数据库的查询压力，提升数据读取速度。

 **二级缓存策略**：结合本地缓存（如Caffeine）和Redis缓存，实现数据的快速访问和高效管理。  **缓存失效策略**：设置合理的缓存过期时间和更新机制，确保数据的实时性和准确性。

**7.消息队列与异步处理**

 **解耦服务模块**：通过**RocketMQ**实现服务之间的异步通信，降低服务之间的耦合度，提升系统的灵 活性。

 **高吞吐量与可靠性**：选择**RocketMQ**作为消息队列，支持高吞吐量和高可靠性的消息传递，确保关 键业务消息的可靠传输和处理。

 **消息幂等性**：设计消息处理机制，确保消息的幂等性，避免重复处理导致的数据不一致。

**（三） 系统附加说明**

**1.软件的运行与开发环境**

WebStorm ，Vue3 ， npm10.9.0 ， node.js v22.11.0 IntelliJ IDEA 2024.2.1 (Ultimate Edition)

docker

MySQL 8.0 Redis

Nacos

Minio

Cassandra

zookeeper:3.5.6

**2.软件的*安装与卸载方*法**



**（四） 本系统的实际开发记录 2024/11/15**

 完成组队，确定成员分工：成员A负责前端，成员B负责后端，成员C负责系统架构和DDD理论指 导。

**2024/11/17**

 初步任务分配与讨论：讨论项目功能需求，确立开发目标，确定使用DDD理论进行开发，划分系统 核心领域。

**2024/11/20**

 完成前端框架和后端基础框架搭建：前端使用Vue3，后端使用Spring Boot，数据库选择MySQL， 配置基本路由、页面和后端接口结构。确定接口数量

**2024/11/22**

 完成用户登录模块：实现用户注册、登录、身份验证功能，后端完成用户信息存储与会话管理，前 端完成登录和注册页面。

**2024/11/25**

 完成笔记发布与管理模块：用户可以创建、编辑、删除笔记，后端API开发完成，前端页面设计并实 现。

**2024/11/30**

 完成点赞与收藏功能开发：实现点赞、收藏操作，数据库模型调整，增加点赞和收藏数据表。

**2024/12/05**

 完成评论功能开发：用户可以评论笔记，支持查看和删除评论，后端接口开发完成，前端完成评论 列表和输入框。

**2024/12/10**

 完成关注与粉丝功能：用户可以关注其他用户，查看粉丝和关注列表，后端实现关注关系管理，前 端展示界面完成。

**2024/12/12**

 整体功能调试与问题修复：整合各模块功能，测试并修复前后端交互问题，调整UI，优化用户体 验。

**2024/12/15**

 完成前后端接口的性能优化：缓存常用API，减少数据库访问频次，优化前端加载速度。

**2024/12/18**

 完成部署环境配置与测试：配置开发、测试和生产环境，完成Docker容器化部署，测试部署过程确 保各环境正常运行。

**2024/12/25**

 完成系统安全性测试与加固。

**2024/12/27**

 编写项目文档与总结：

（2）给出项目组成员分工列表

1.李治衡 1120223468：总体项目规划，后端设计开发，系统部署。  
2.张一晨：前端代码编写，文档撰写及整合，需求分析

3.尉坤 ：前端代码编写，文档撰写，图片选取。