

本科毕业论文（设计）

题目  LuminaryBlogs博客系统的设计与实现

学 院 商贸学院

专 业 信息管理与信息系统

年 级 2020级

学 号 222020603063002

姓 名 陈国飞

指 导 教 师 肖兴江

成 绩

2024年5月21日

目  录

[摘要 1](#_Toc134602173)

[Abstract 1](#_Toc134602174)

[第1章 绪论 2](#_Toc134602175)

[1.1 研究背景及意义 2](#_Toc134602176)

[1.2 研究现状 3](#_Toc134602177)

[1.3 研究内容 4](#_Toc134602178)

[第2章 系统相关理论与技术 5](#_Toc134602179)

[2.1 JSON Schema 5](#_Toc134602180)

[2.2 关键技术选择 6](#_Toc134602181)

[2.2.1 前端技术选型 6](#_Toc134602182)

[2.2.2 后端技术选型 6](#_Toc134602183)

[2.3 数据库选择 7](#_Toc134602184)

[2.4 其他工具 8](#_Toc134602185)

[第3章 系统分析 9](#_Toc134602186)

[3.1 可行性分析 9](#_Toc134602187)

[3.2 系统设计目标 10](#_Toc134602188)

[3.3 系统功能需求分析 10](#_Toc134602189)

[3.4 系统性能需求 11](#_Toc134602190)

[第4章 系统设计 12](#_Toc134602191)

[4.1 系统功能模块设计 12](#_Toc134602192)

[4.1.1 登录注册模块 12](#_Toc134602193)

[4.1.2 店铺装修模块 13](#_Toc134602194)

[4.1.3 页面模块 14](#_Toc134602195)

[4.1.4 商品模块 15](#_Toc134602196)

[4.1.5 自定义属性控制面板 16](#_Toc134602197)

[4.1.6 模板市场 16](#_Toc134602198)

[4.2 数据库设计 17](#_Toc134602199)

[4.2.1 数据库概念结构设计 17](#_Toc134602200)

[4.2.2 数据库逻辑结构设计 18](#_Toc134602201)

[第5章 详细系统的实现 22](#_Toc134602202)

[5.1 登录注册模块实现 22](#_Toc134602203)

[5.2 店铺装修模块实现 22](#_Toc134602204)

[5.3 页面模块实现 23](#_Toc134602205)

[5.4 商品模块实现 24](#_Toc134602206)

[5.5 自定义属性控制面板实现 25](#_Toc134602207)

[5.6 模板市场实现 26](#_Toc134602208)

[第6章 系统测试 28](#_Toc134602209)

[6.1 测试方案 28](#_Toc134602210)

[6.2 功能测试 29](#_Toc134602211)

[6.2.1 用户注册测试 29](#_Toc134602212)

[6.2.2 用户登录测试 30](#_Toc134602213)

[6.3 兼容性测试 31](#_Toc134602214)

[6.4 安全测试 31](#_Toc134602215)

[6.4.1 认证和授权测试 31](#_Toc134602216)

[6.4.2 安全漏洞测试 32](#_Toc134602217)

[第7章 总结与展望 33](#_Toc134602218)

[7.1 总结 33](#_Toc134602219)

[7.2 未来工作展望 34](#_Toc134602220)

[参考文献 36](#_Toc134602221)

[致谢 37](#_Toc134602222)

LuminaryBlogs博客系统的设计与实现

陈国飞

西南大学商贸学院，重庆荣昌 402460

**摘要**：随着互联网的快速发展和普及，博客成为了人们获取信息、分享经验和表达观点的重要平台。然而，当前存在大量博客内容的质量参差不齐，缺乏权威性和吸引力，使用户难以找到真正有价值的内容。为了解决这一问题并提供一个具有权威性和吸引力的博客平台，本论文提出了LuminaryBlogs博客系统。

该系统旨在提供一个具有权威性和吸引力的博客平台，为用户提供精确的内容，致力于满足用户对各种技术和非技术领域博客的浏览和学习需求，并为用户提供创作和分享内容的功能。系统设计着重关注用户体验和界面设计，以提供简洁、直观的博客浏览和阅读体验。此外，系统采用智能推荐算法，根据用户的偏好和浏览信息，为用户推荐相关博客内容。

该系统采用Kotlin语言，选用MySQL作为数据库并使用Exposed操控数据库的增删改查，结合Intellij IDEA开发工具和目前比较新的后端开发技术Ktor开发系统后台，并结合Android Studio和Jetpack Compose开发前端应用。

**关键词：**权威性；智能推荐。

**Design and Implementation of LuminaryBlogs Blog System**

Chen Guofei

Business College, Southwest University, Chongqing 402460, PR China

**Abstract:** With the rapid development and popularization of the Internet, blog has become an important platform for people to obtain information, share experience and express views. However, there is a large number of blog content with uneven quality, lack of authority and attractiveness, and it is difficult for users to find truly valuable content. In order to solve this problem and provide an authoritative and attractive blogging platform, the LuminaryBlogs blogging system is proposed in this paper.

The system aims to provide an authoritative and attractive blogging platform that provides users with precise content, is dedicated to meeting users' browsing and learning needs for blogs in various technical and non-technical fields, and provides users with the ability to create and share content. The system design focuses on user experience and interface design to provide a simple and intuitive blog browsing and reading experience. In addition, the system uses intelligent recommendation algorithm to recommend relevant blog content for users according to their preferences and browsing information.

The system adopts Kotlin language, chooses MySQL as the database and uses Exposed to control the database addition, deletion, correction and check, combined with Intellij IDEA development tools and the current relatively new back-end development technology Ktor development system background. It combines Android Studio and Jetpack Compose to develop front-end applications.

**Keywords:** Authority; Intelligent recommendation.

# 第1章 绪论

在这个互联网时代，怎么快速高效的获取自己想要的文章成为了一个难题，互联网上充斥着各种各样的文章，这些文章中有的不具备权威性很容易误导用户，在搜索的过程中也会遇到与搜索无关的内容，例如广告，弹窗等，这些都大大降低了用户学习和工作的效率。LuminaryBlogs旨在为用户提供一个全面且高质量的博客内容平台。通过该系统，用户可以方便地浏览和学习各种技术和非技术领域的博客内容，并深入了解和探索新的技术和知识。同时，LuminaryBlogs也鼓励用户积极参与到博客创作和分享中，使用户能够创作和发布自己的博客内容，记录个人的生活、学习经历或技术讲解等。

通过本论文的研究和实现，LuminaryBlogs博客系统将为用户提供一个优质、权威且吸引人的博客平台，促进用户之间的知识共享、交流和互动，满足用户对高质量博客内容的需求，并推动博客领域的进一步发展和创新。

## 1.1 研究背景及意义

本研究的意义在于满足用户对高质量、权威性博客内容的需求，并提供一个优质的博客平台。LuminaryBlogs系统将为用户提供一个可靠的知识获取和分享平台，帮助用户获取有价值的信息，拓展知识领域，促进个人和专业发展。此外，系统的智能推荐功能将根据用户的兴趣和偏好，提供个性化的博客推荐，使用户能够更高效地获取感兴趣的内容。通过这些努力，LuminaryBlogs系统有望推动博客领域的进一步发展，促进知识的共享和交流，为用户带来更丰富、有价值的信息体验。

## 1.2 研究现状

当前博客系统领域的一些主要研究现状：

* 内容质量筛选：许多研究关注如何筛选和提供高质量的博客内容。其中一些研究集中于通过用户评价、专家审核或机器学习算法等方式，对博客进行质量评估和筛选，以提供可靠和有价值的信息资源。
* 用户体验改进：用户体验是博客系统设计中的重要方面。研究者们致力于改进界面设计、导航结构和交互方式，以提供更直观、易用和愉悦的用户体验。一些研究还关注个性化定制和推荐功能，以满足用户的个性化需求。
* 智能推荐系统：为了帮助用户发现感兴趣的博客内容，智能推荐系统在博客领域得到广泛应用。这些系统基于用户的浏览历史、兴趣偏好和社交网络等信息，采用协同过滤、内容分析和机器学习等技术，为用户提供个性化的博客推荐和推荐排名。
* 博客分类和标签：为了方便用户浏览和检索博客内容，研究人员关注如何进行博客分类和标签。这些研究包括基于文本挖掘和自然语言处理技术的自动分类方法，以及用户协同标注和社区贡献的标签方法，以提高博客内容的组织和发现效率。
* 社交互动和评论：博客系统也注重用户之间的社交互动和评论功能。研究者们探索如何设计有效的评论系统、点赞和分享功能，以促进用户之间的交流和知识共享。

尽管已经有了许多有关博客系统的研究，但仍存在一些挑战和改进空间。例如，提供更精确的内容质量筛选和评估算法，进一步改进用户体验和界面设计，提供更准确和个性化的推荐服务等。本研究旨在在这些方面做出贡献，开发出一种具有权威性和吸引力的博客系统——LuminaryBlogs，以满足用户对高质量博客内容的需求，并提供优质的用户体验和个性化服务。

## 1.3 研究内容

该系统的研究内容主要包括以下几个方面：

（1）博客内容质量筛选和权威性：研究如何通过评估和筛选机制，提供具有高质量和权威性的博客内容。这可能涉及到设计评估指标、制定筛选算法，以及吸引专业作者和内容创作者来发布优质的博客内容。

（2）用户个性化推荐：研究如何利用用户的兴趣、浏览历史和行为数据，应用智能推荐算法，为用户提供个性化的博客内容推荐。这可能涉及到协同过滤、内容分析和机器学习等技术，以提高推荐的准确性和用户满意度。

（3）博客排序和分类：研究如何设计排序算法和博客分类方法，以便用户可以根据自己的偏好和需求对博客进行排序和筛选。这可能包括基于关键词和标签的分类方法，以及根据用户反馈和评价进行的排序算法。

（4）用户体验和界面设计：研究如何设计简洁、直观且易用的用户界面，以提供良好的用户体验。这可能包括界面布局、导航结构、交互方式和视觉设计等方面的研究和优化。

（5）社交互动和评论功能：研究如何设计有效的社交互动和评论功能，以促进用户之间的交流和知识共享。这可能包括评论系统的设计、点赞和分享功能的实现，以及用户之间的互动和社区建设等方面的研究。

（6）通过以上研究内容的探索和实现，LuminaryBlogs博客系统旨在提供一个全面、高质量和个性化的博客内容平台，满足用户对高质量博客内容的需求，并提供优质的用户体验和社交互动功能。

# 第2章 系统相关理论与技术

## 2.1 Ktor

Ktor是一个用于轻松构建异步服务器端和客户端应用程序的框架。在本系统中，后台采用Ktor Server编写，前端网络请求框架使用Ktor Client实现。

## 2.2 Kotlin

Kotlin 是一种现代化、静态类型的编程语言，由 JetBrains 公司于2011年开发并在2016年正式发布。它被设计成与 Java 互操作并运行在 Java 虚拟机（JVM）上，因此可以与现有的 Java 代码和库很好地集成。Kotlin 的出现为开发者提供了一种现代化、强大且易用的编程语言选择，它广泛应用于 Android 开发、后端开发、Web 开发以及其他领域。其优雅的语法、丰富的功能和与现有技术的无缝集成使得 Kotlin 成为一个备受青睐的编程语言。

## 2.3 Jetpack Compose

Jetpack Compose 是一种用于构建 Android 用户界面的现代化 UI 工具包。它是由谷歌开发的，旨在简化和改进 Android 应用程序的 UI 开发流程。

相对于传统的基于 XML 的布局系统，Jetpack Compose 提供了一种声明式的方式来构建用户界面，使用 Kotlin 语言编写。它采用了函数式编程的思想，通过组合函数和可组合的 UI 元素来构建界面。

Jetpack Compose 是 Android 开发的未来趋势，它为开发者带来了更高效、更灵活和更愉悦的用户界面开发体验。尽管它仍处于不断演进的阶段，但已经受到广大开发者的关注和采用，并且逐渐成为 Android 应用程序开发的主流选择。

## 2.4 Exposed

Exposed是JDBC驱动程序之上的一个轻量级SQL库，Exposed有两种类型的数据库访问：类型安全的SQL wrapping DSL和轻量级数据访问对象（DAO）,本系统采用的是SQL wrapping DSL。 Exposed可以用来模拟各种数据库引擎，在不依赖任何特定数据库引擎的情况下构建应用程序，并在它们之间切换，只需很少或根本不需要更改。

## 2.5 Material Design

Material Design 是由谷歌推出的一种用户界面设计语言和设计系统，旨在为移动应用、Web 应用和其他数字平台提供一致、直观和美观的设计风格。Material Design 的设计原则基于真实世界的物理材料和纸张，通过阴影、深度、动画和颜色等元素来创造出具有层次感和逼真效果的界面。

## 2.6 后端技术选型

后端技术选型为Kotlin+Ktor Server+Exposed，这是一种流行且强大的组合，适用于构建现代化、高性能的 Web 后端应用程序。

（1）Kotlin：作为一种现代化的编程语言，Kotlin 提供了许多优势，如简洁性、安全性和与 Java 的互操作性。它具有丰富的特性和函数式编程的支持，使得代码编写更加简洁、易读和易于维护。

（2）Ktor：Ktor 是一个轻量级、异步、非阻塞的 Web 框架，专为 Kotlin 设计。它基于协程（Coroutines）的概念，提供了简洁而强大的 API，使开发者能够快速构建高性能的 Web 服务。Ktor 支持各种常见的 Web 服务器和协议，如 HTTP、WebSocket 等。本系统后台中的安全认证JWT、Session等都是基于Ktor。

（3）Exposed：Exposed 是 Kotlin 的一种简洁且类型安全的 SQL 查询和数据库访问库。它提供了直观的 DSL（领域特定语言），使得数据库操作变得简单和易于管理。Exposed 能够轻松地与常见的关系型数据库进行集成，如 PostgreSQL、MySQL 等。

Kotlin+Ktor+Exposed 的技术组合提供了一种现代化、高效和安全的后端开发方式，适用于构建各种规模的 Web 应用程序。它们的配合能够提供良好的开发体验和性能表现，并为开发者带来更高效的开发和维护工作流程。

## 2.7 前端技术选型

前端技术选型为 Kotlin+Ktor Client+Jetpack Compose+Material Design

（1）Jetpack Compose：Jetpack Compose 是谷歌推出的一种声明式 UI 工具包，用于构建现代化的 Android 应用程序。使用 Kotlin 语言编写，Jetpack Compose 提供了一种简洁、直观和响应式的方式来构建用户界面。它基于函数式编程的概念，通过组合可重用的 UI 组件来构建复杂的界面。

（2）Material Design：Material Design 是谷歌推出的一种用户界面设计语言和设计系统，提供一致、直观和美观的设计风格。在前端开发中，Material Design 与 Jetpack Compose 结合使用，可以实现现代化和美观的用户界面。Material Design 提供了丰富的组件、颜色和样式，使得开发者能够轻松地创建具有一致性和吸引力的界面。

Kotlin+Ktor+Jetpack Compose+Material Design 的技术组合提供了一种现代化、高效和美观的前端开发方式。它们的配合能够提供良好的开发体验、可维护性和用户体验，并使开发者能够构建出令人印象深刻的移动应用。

# 第3章 系统分析

## 3.1 可行性分析

（1）技术可行性：该系统采用Ktor框架、Exposed、Jetpack Compose、Kotlin等先进技术开发。Ktor是目前比较流行的后端开发框架，同时它也是客户端的网络库，可以替换目前比较流行的Retrofit、OkHttp等网络库，它集成了JWT、Session等安全认证机制，结合Kotlin提供了简单易用的api，对开发人员很友好，有着简单、易用、高效等特点。Exposed是 Kotlin 的一种简洁且类型安全的 SQL 查询和数据库访问库，它可以屏蔽各种数据库之间的差异，即底层数据库可以在任意数据库如MySQL、MongoDB之间切换。Jetpack Compose是现代化的Android UI工具包，以声明式的方法描述UI，简化UI开发。这些高效简洁易用技术的组合，使得该系统开发的难度、复杂度大大下降。

（2）经济可行性：该系统使用到的技术都特别简单易用和高效，从而降低了开发的成本，同时也降低了后期的升级维护成本。该项目完成后能给用户带来一种全新的博客浏览体验，权威的博客内容、精确的内容推送等可以吸引大量用户，实现平台收益。

（3）操作可行性：该系统的UI设计简洁明了，在使用时可以降低用户的心智负担，使用户可以专注于浏览自己感兴趣的博客。界面组件、颜色搭配、字体大小等遵循了Material Design的设计风格，可以提高用户的使用体验，提高用户留存率。

## 3.2 系统设计目标

（1）功能性：系统能够实现所有预定义的功能需求，以支持用户完成特定的任务，包括登录、注册、写作、浏览博客、个人信息设置等。此外本系统还提供直观、易用的用户界面，确保用户能够高效地与系统互动，同时系统还需要具有足够的灵活性，以便在未来可以添加或修改功能。

（2）可靠性：系统能在预期的负载下稳定运行，不会频繁崩溃或失去相应；系统具备错误处理能力，能够在发生错误时继续运行，最小化系统停机时间。

（3）可维护性：通过模块化设计，使得系统易于理解、修改、测试和 维护；提供清晰的文档，包括模块设计、代码注释和用户手册，以便于维护和更新。

（4）安全性：系统确保敏感数据的安全，对用户敏感数据使用加密方式存储；系统能够进行身份认证和授权，确保用户只能访问他们被授权的数据和功能。

## 3.3 系统功能需求分析

本系统的功能包括登录注册、浏览博客、收藏、评论、点赞、创作和编辑博客、搜索博客或用户、修改个人信息、根据用户喜好推荐博客等等，所以该系统应该具备的功能如下：

1. 用户账户管理：提供登录、注册和退出登录功能，登录注册时提供用户协议和隐私政策供用户阅读并同意。
2. 修改个人信息：用户可以在账户设置中修改个人资料，如头像、昵称、背景图等。
3. 修改系统主题：用户可以根据喜好修改主题为日间或者夜间模式，或者跟随系统。
4. 博客浏览与交互：用户可以查看博客列表以及博客详细内容，包括博客标题、摘要、作者、发布时间和阅读量等信息，支持滑动或点击加载更多博客。另外，还提供了错误情况下的交互内容，例如没网络时显示无网络插图并提醒用户开启网络，列表为空时提示用户暂时无内容的插图等等。
5. 收藏博客：用户可以收藏喜欢的博客文章，并在收藏列表查看。
6. 评论博客：用户可以对博客文章发表评论，评论区显示评论者名称、评论时间和内容。
7. 点赞博客：用户可以为博客文章点赞，显示博客的总点赞数。
8. 博客创作与管理：
9. 用户可以创建新的博客文章，包括添加标题、内容、图片、链接、标签等。
10. 用户可以编辑自己发布的博客文章，例如修改博客可见范围或者删除博客等。
11. 用户可以查看和管理自己发布的博客列表。
12. 搜索功能：用户可以通过关键词搜索博客和用户。
13. 个性化推荐：根据用户的阅读历史、朋友的点赞收藏和评论信息，推荐相关的博客文章。
14. 安全与隐私：确保用户数据的安全存储和传输，使用加密技术和认证授权保护用户隐私。

## 3.4 系统性能需求

性能需求方面，本系统在响应时间、可伸缩性、容错性、兼容性、数据完整性方面进行把控，具体的指标及要求如表3.1所示：

表3.1 系统性能需求表

|  |  |
| --- | --- |
| 性能需求指标 | 具体要求 |
| 响应时间 | 本系统冷启动耗时在1.3s以内；确保主线程只处理UI相关的操作，快速响应用户的点击、滑动等事件。 |
| 可伸缩性 | 能够根据用户量自动水平扩展服务，可以应对突发的流量高峰。 |
| 容错性 | 本系统后端使用Ktor开发，可以保证服务器运行期间不会因任何原因导致服务器崩溃，同时客户端网络请求及处理都做了异常捕获，根据请求结果显示相应内容，确保应用不会崩溃。 |
| 兼容性 | 本系统兼容Android API 级别 21（Android 5.0）及以上版本，覆盖95%以上的Android系统。 |
| 数据完整性 | 本系统借助Exposed实现了事务完整性，确保数据的一致性和准确性 |

# 第4章 系统设计

系统设计是软件工程中的一个阶段，它涉及到创建系统的架构、组件、模块、接口和数据的过程。在这个过程中，本系统根据之前如需求分析阶段收集到的信息，来定义系统的工作方式和组织结构，并产出系统的详细设计方案。

## 4.1 系统功能模块设计

系统功能模块设计是在软件开发过程中对系统的功能进行结构化分解的过程。它涉及将系统的整体功能需求分解为多个较小、更易管理的部分，即模块。这些模块通常围绕特定的功能、业务逻辑或服务进行组织。模块化设计有助于提高系统的可维护性、可扩展性和复用性。该系统从功能上被划分为9个子模块：登录注册、首页文章列表、浏览博客详情、编辑创建、搜索、消息中心、设置、个人中心、粉丝及朋友中心，系统功能结构如图4.1所示。

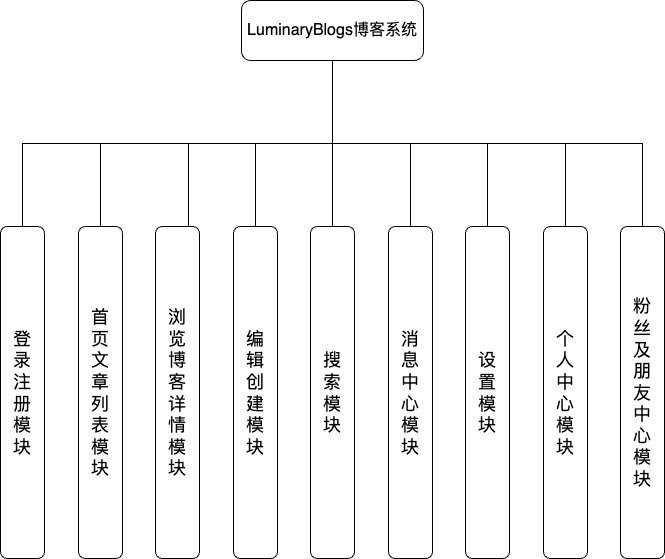


图4.1 系统功能结构图

Fig. 4.1 System function structure diagram

### 4.1.1 登录注册模块

登录注册模块负责管理用户的访问权限和身份验证，在登录中需要用户提供账号和密码并同意隐私政策，当后台验证通过后登录成功，否则登录失败。注册时用户需要提供用户名和密码，且用户名不能与现有用户的重复，同意隐私政策后方可注册，注册成功则跳转登录页面。用户登录注册流程如图4.2所示。

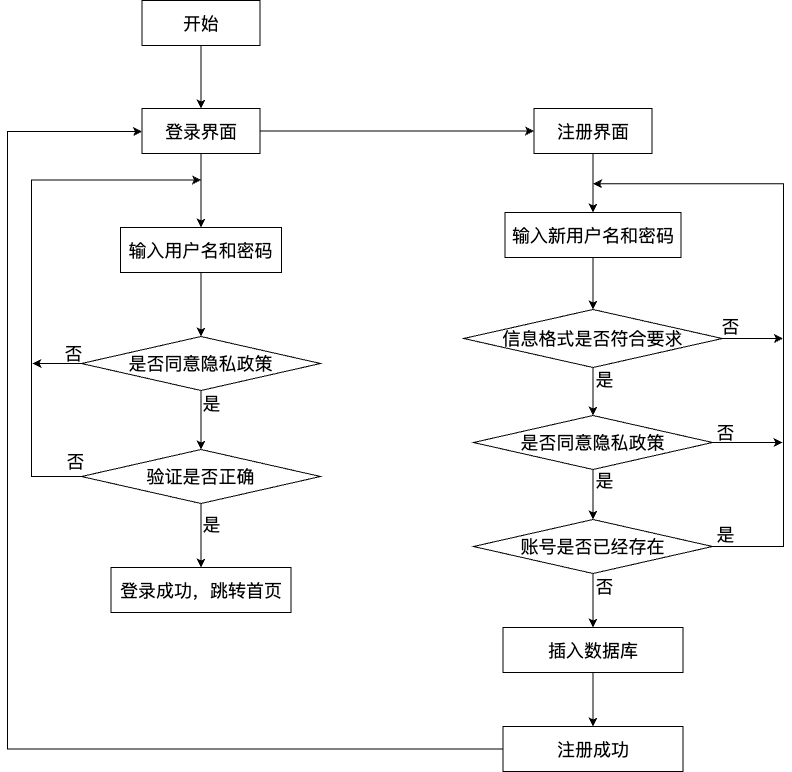


图4.2登录注册流程图

Fig. 4.2 Logon registration flowchart

### 4.1.2 首页文章列表模块

一个应用的首页对于用户留存率和整体体验感至关重要。作为用户的第一印象，首页必须迅速传达应用的核心功能和价值主张，同时提供直观、易用的导航，确保用户能够无障碍地访问他们感兴趣的内容或功能。一个设计精良的首页能够有效吸引用户深入探索，提高用户的参与度和满意度，从而提升用户留存率。此外，首页的加载速度、响应性和个性化元素也是提升用户体验和留存的关键因素，为了满足加载和响应速度，本系统所有含有列表的页面都采用分页加载，实现按需加载的同时还能节省资源、提高响应速度和屏幕渲染速度。本系统首页的主要功能有：

（1）默认推荐文章列表：系统根据用户喜好或者朋友浏览、点赞、收藏、评论过的文章进行智能推送，提供用户感兴趣的内容。

（2）全部文章列表：展示系统的全部文章，默认按发布先后时间进行排序

（3）朋友文章列表：展示朋友的文章，朋友为相互关注的用户。

（4）工具栏和导航栏配置：工具栏提供新建文章和搜索文章及用户入口；导航栏提供首页、消息中心、个人中心入口。

首页文章列表模块功能结构如图4.3所示。

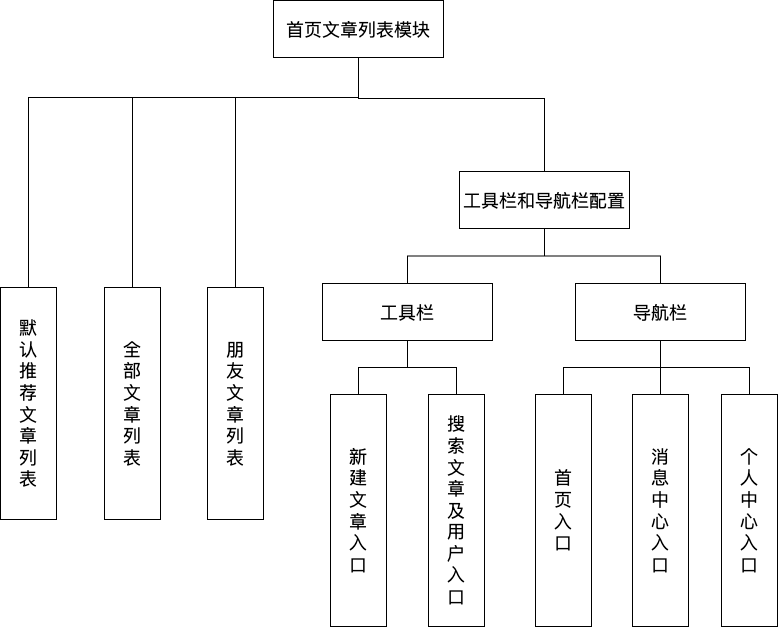


图4.3 首页文章列表功能结构图

Fig.4.3 Home article list function structure chart

### 4.1.3 浏览博客详情模块

博客详情页面是用户获取内容和深入了解主题的关键接触点。一个设计良好的博客详情页面不仅能够提供丰富、有价值的内容，还能通过清晰的格式和舒适的阅读体验吸引用户停留更长时间。此外，页面上的互动元素，如评论区、分享按钮，可以鼓励用户参与社交互动，增加用户的参与度和忠诚度。本系统的浏览博客详情模块提供了如下功能：

1. 浏览文章内容：包括标题、正文、作者信息。
2. 更多：包括分享文章、复制链接、权限设置、删除、编辑。
3. 互动区域：包括文章评论、写评论、点赞、收藏。

浏览博客详情功能结构如图4.4所示。

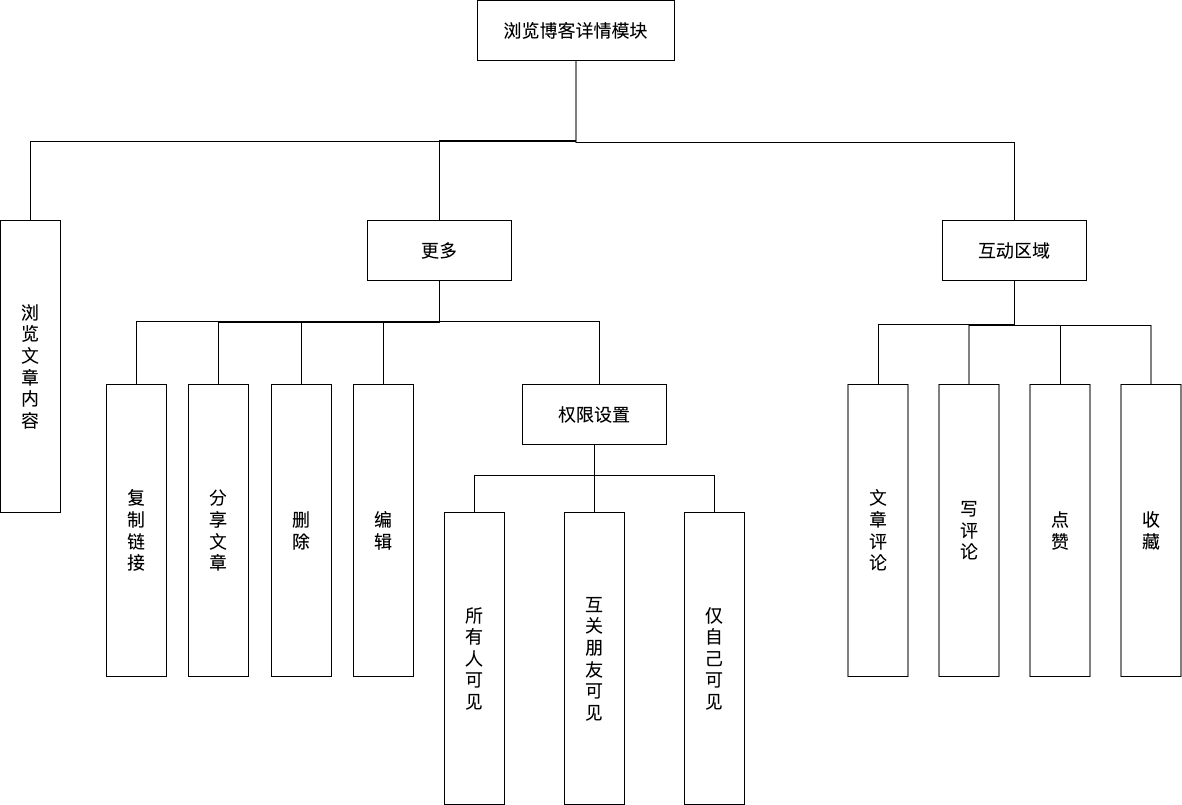


图4.4 浏览博客详情功能结构图

Fig.4.4 Browse the blog Details feature chart

### 4.1.4 编辑创建模块

编辑和创建文章页面是博客系统中至关重要的组成部分，因为它们直接关系到用户生成内容的能力和体验。本系统编辑和创建博客页面提供了一个直观、易用且功能丰富的编辑器，让用户能够轻松地格式化文本、插入媒体和链接，以及预览他们的文章。良好的用户体验可以激发用户的创作热情，本系统的编辑器支持目前流行的Markdown格式，降低了技术门槛，使所有用户都能够表达自己的想法和分享知识。此外，本系统支持草稿保存、自动保存草稿编辑，能够增强用户的信心，让他们在创作过程中感到安心。

本系统的编辑和创建模块提供了如下功能：

1. 编辑页面：支持用户使用Markdown格式编辑文章内容，提供常用的一级二级等标题、粗体、斜体等常用功能，此外提供了插入图片、视屏、链接功能。
2. 预览页面：预览用户编辑的文章内容。
3. 发布设置：管理标签，用户可以添加标签或使用历史标签；设置文章可见范围；上传文章封面。

编辑创建模块功能结构如图4.5所示。

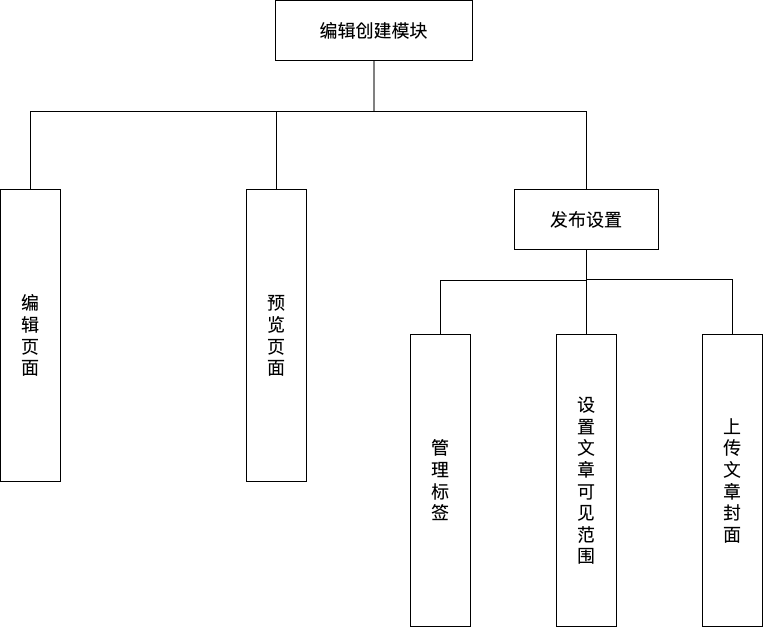


图4.5 编辑创建模块功能结构图

Fig.4.5 Edit and create module function structure diagram

### 4.1.5 搜索模块

搜索功能对用户极其重要，本系统提供了快速定位和访问文章、用户的途径。良好的搜索体验可以显著提高用户满意度，增加用户粘性。

本系统搜索模块提供了如下功能：

1. 搜索文章：根据输入关键字返回相关的文章，关键字可以匹配文章标题、内容和标签。
2. 搜索用户：根据输入的关键字返回相关的用户，根据用户名进行匹配。

搜索模块功能结构如图4.6所示。

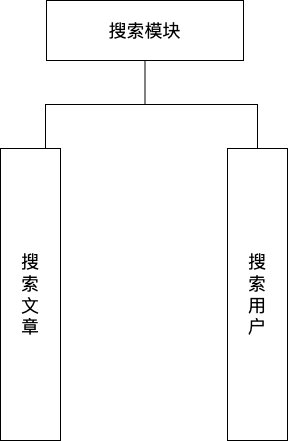


图4.6 搜索模块功能结构图

Fig.4.6 Search module function structure diagram

### 4.1.6 消息中心模块

本系统消息中心允许用户接收到有关新评论、点赞、关注提醒，从而增强用户的参与度和互动体验。通过即时反馈，用户可以迅速回应社区内的互动，建立起与其他用户之间的联系。

本系统的消息中心模块提供了如下功能：

1. 评论页面：显示用户收到的评论或者用户对文章的评论。
2. 点赞通知：显示用户收到的点赞。
3. 关注通知：显示用户收到的关注。

消息中心模块功能结构如图4.7所示。

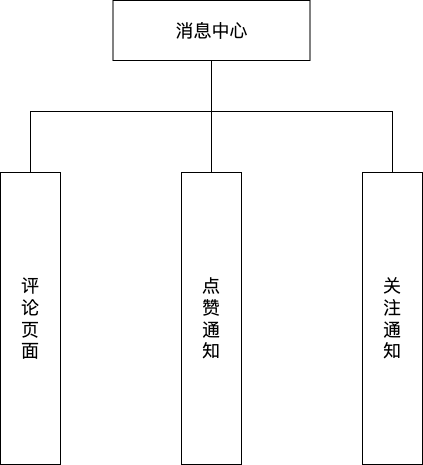


图4.7 消息中心功能结构图

Fig.4.7 Browse the blog Details feature chart

### 4.1.7 设置模块

本系统中用户可以通过设置模块调整账户信息，如更新个人资料、修改主题、查看隐私政策、退出登录等。良好的设置模块设计可以提升用户的整体满意度，促进更深层次的用户参与。

本系统的编辑和创建模块提供了如下功能：

1. 修改主题：用户可以修改主题为深色模式、日间模式、跟随系统。
2. 修改个人资料：用户可以修改头像、北京简介、生日等个人资料。
3. 隐私政策：用户可以查看本系统的隐私政策。

设置模块功能结构如图4.8所示。

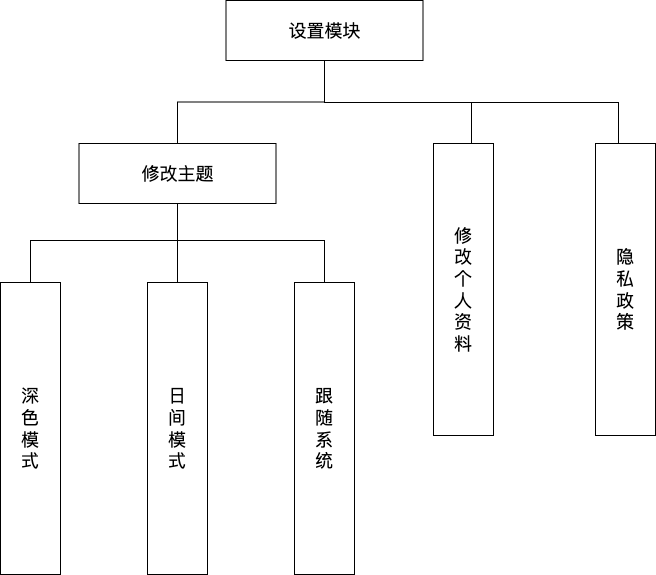


图4.8 设置模块功能结构图

Fig.4.8 Setting module function chart

### 4.1.8 个人中心模块

本系统的个人中心是用户管理自身博客和个人信息的枢纽。它使用户能够轻松查看和编辑个人资料，管理发布的博文，追踪和响应评论，查看和修改草稿内容，以及查看收到的点赞和关注以及朋友和粉丝数量。

个人中心模块提供了如下功能：

1. 个人数据：在个人中心，用户可以看到自己的获赞数量、朋友数量、关注人数和粉丝人数。
2. 文章数据

* 作品：用户可以查看自己公开发布的作品。
* 私密：供用户查看自己的私密文章，其他用户无查看权限。
* 收藏：用户可以查看自己收藏的文章。
* 喜欢：用户可以查看自己点过赞的文章。

1. 草稿箱：用户可以点击进入草稿页面，继续编辑草稿、删除草稿。

个人中心模块功能结构如图4.9所示。

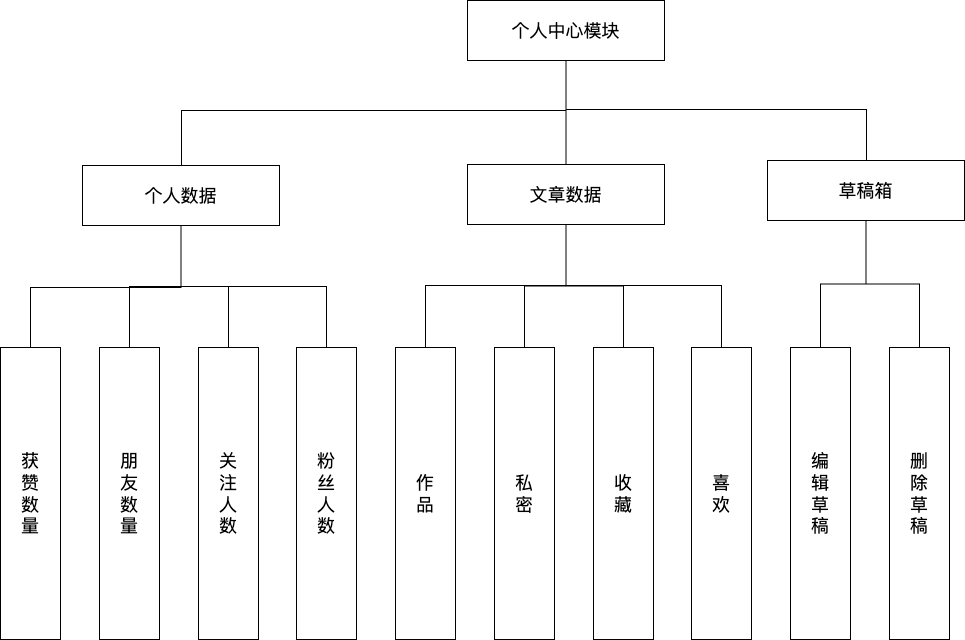


图4.9 个人中心模块功能结构图

Fig.4.9 Individual central module function structure diagram

### 4.1.9 粉丝及朋友中心模块

粉丝及朋友中心允许用户查看和管理自己的社交连接，包括粉丝、关注的人和朋友。通过这个中心，用户可以跟踪谁在关注他们的更新，以及他们自己感兴趣的博主和内容。

粉丝及朋友中心模块提供了如下功能：

1. 查看朋友：朋友是相互关注的用户。
2. 查看关注关注：查看用户关注的用户。
3. 查看粉丝：查看谁关注了该用户。

粉丝及朋友中心功能结构如图4.5所示。

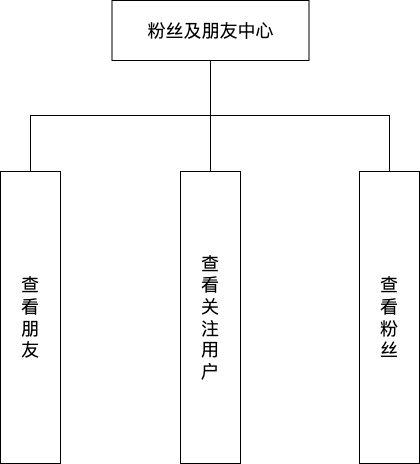


图4.10 粉丝及朋友中心功能结构图

Fig.4.10 Fans and friends center function structure diagram

## 4.2 数据库设计

### 4.2.1 数据库概念结构设计

数据库设计的核心目标是创建一个结构化、高效和可靠的数据存储系统，它能够确保数据的完整性、一致性和安全性。良好的数据库设计应支持快速准确的数据检索，便于用户访问和分析数据。同时，它应具备良好的伸缩性，能够随着数据量的增长和用户需求的变化而灵活扩展。数据库还需要实现有效的数据保护措施，防止未授权访问和数据泄露，确保遵守相关的数据隐私法规。此外，数据库设计还应考虑到性能优化，减少资源消耗，提高操作效率，以保持系统的响应速度和可用性。

本系统主要涉及的实体有用户、文章、收藏、评论、朋友、点赞、浏览文章、浏览时长。主要的实体E-R图如4.11所示。

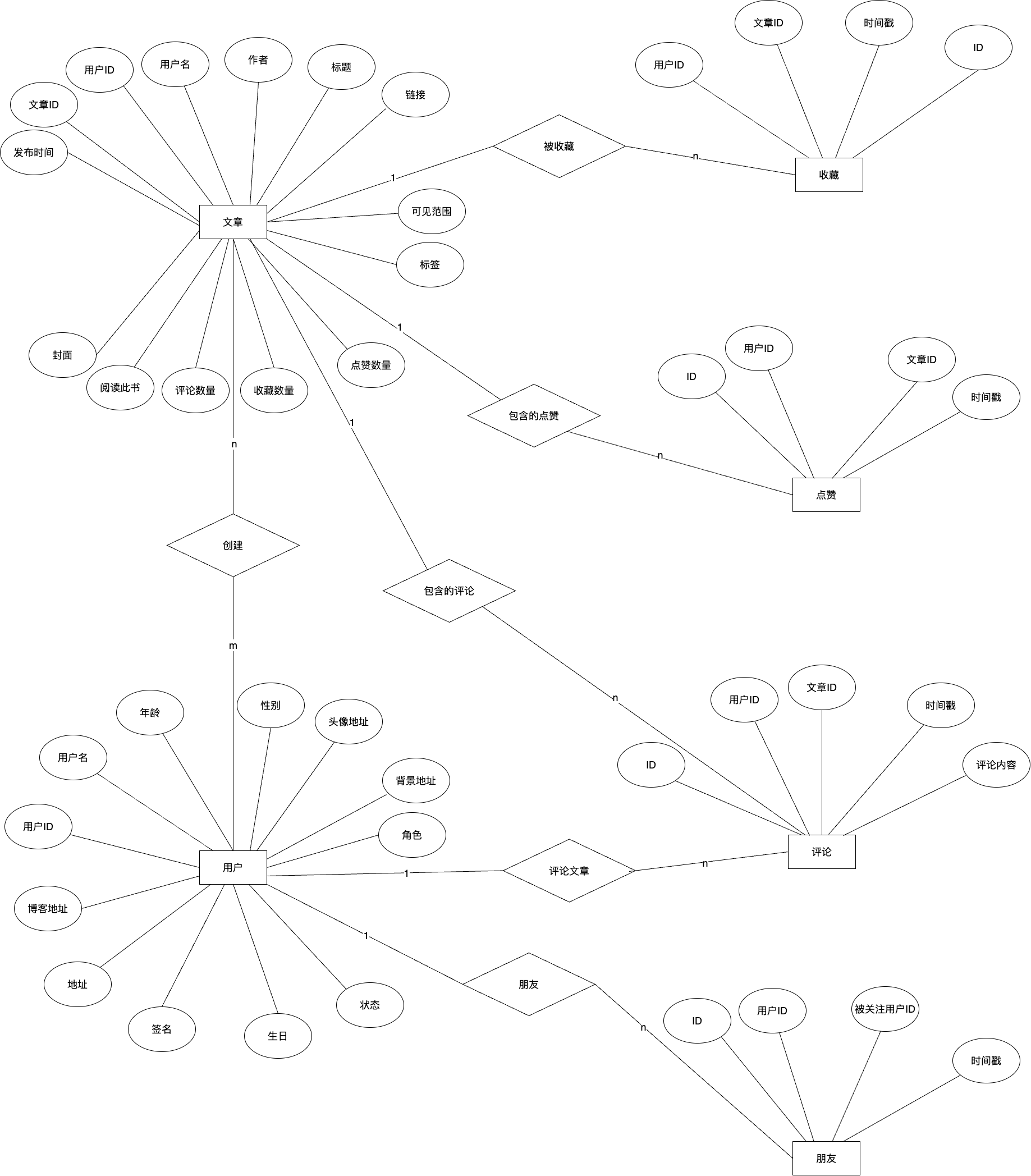


图4.11 主要的实体图

Fig.4.11 Major entity chart

各实体及其属性如下：

（1）用户实体主要用于存储用户相关的信息，包括但不限于用户ID、用户名、密码、头像、签名和其他个人资料。它为系统提供了一个中心化的用户身份和属性管理机制，支持身份验证、权限控制、个性化服务和用户行为追踪等功能，如图4.11所示。

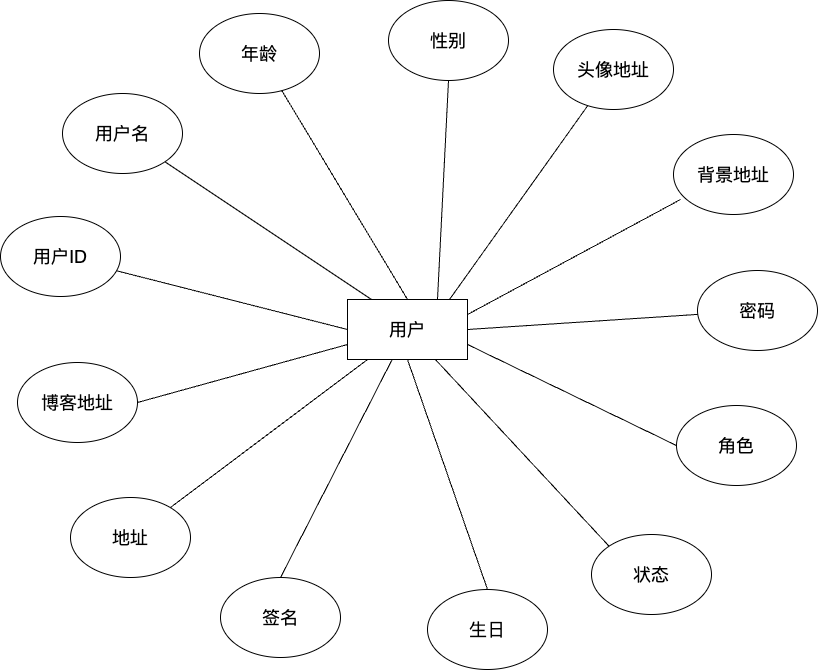


图4.12 用户实体图

Fig.4.12 User entity chart

（2）文章实体存储了文章的相关信息，例如文章ID、标题、链接、内容、作者等，如图4.13所示。

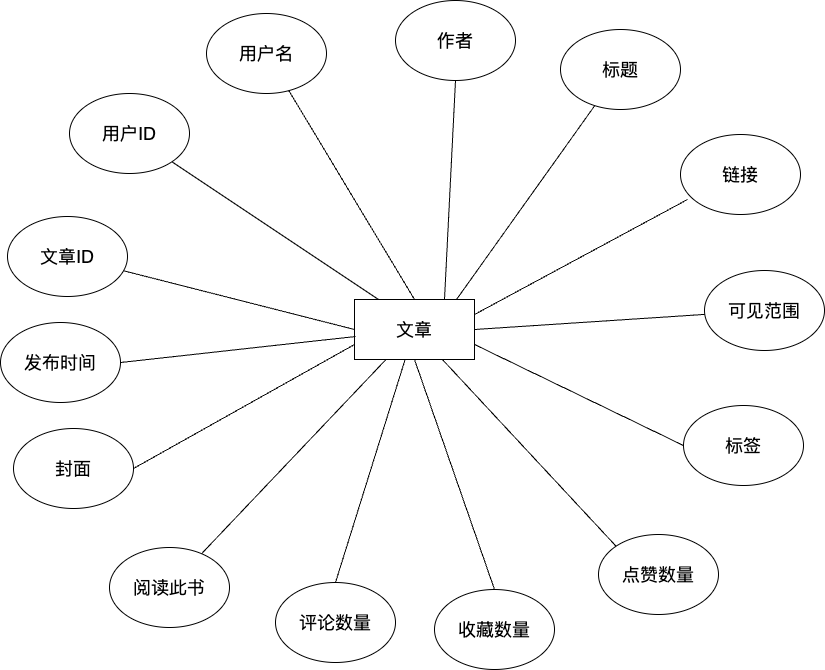


图4.13 文章实体图

Fig.4.13 Article entity chart

1. 收藏实体存储了文章收藏信息，包括谁收藏了该文章以及文章ID等，如图4.14所示。

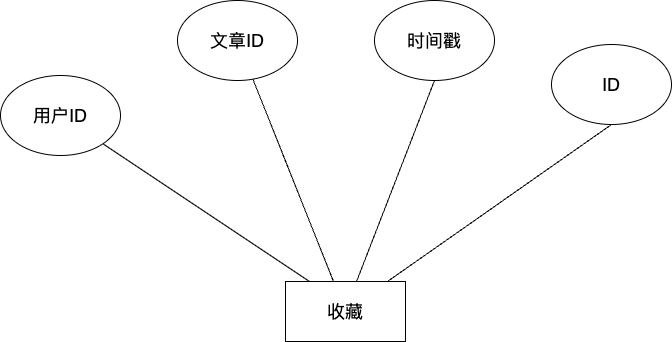


图4.14 收藏实体图

Fig.4.14 Collect entity chart

1. 评论实体存储了评论信息，包括了评论用户，评论文章，评论内容等，如图4.15所示。

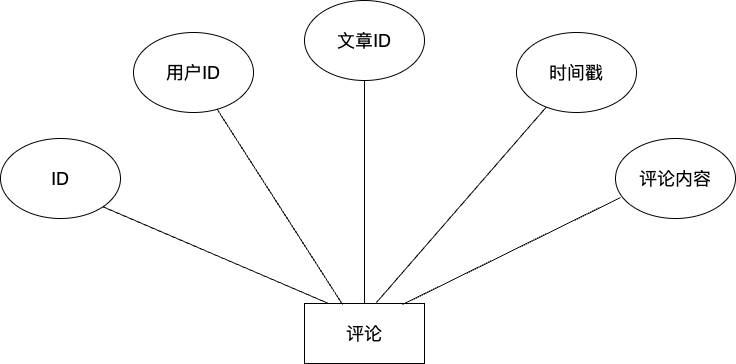


图4.15评论实体图

Fig.4.15 Comment entity chart

1. 朋友实体存储了用户ID、被关注用户ID等信息，如图4.16所示。

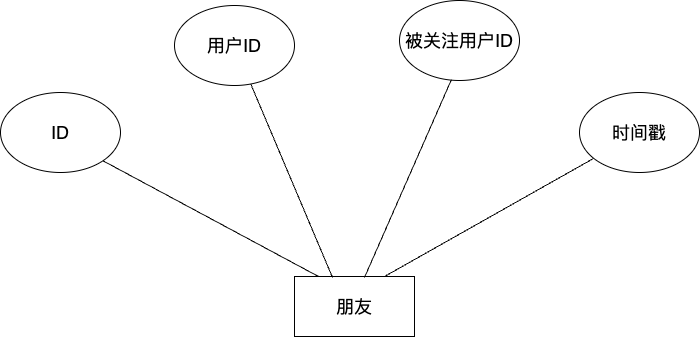


图4.16 朋友实体图

Fig.4.16 Friend entity chart

1. 点赞实体存储了用户ID、文章ID等信息，如图4.17所示。

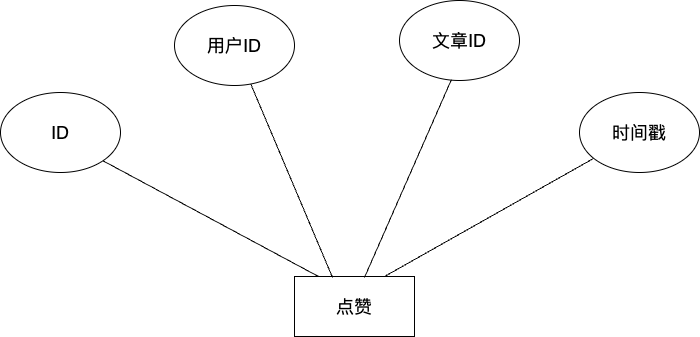


图4.17 点赞实体图

Fig.4.17 Like entity chart

1. 浏览文章实体存储了用户ID、文章ID等信息，如图4.18所示。

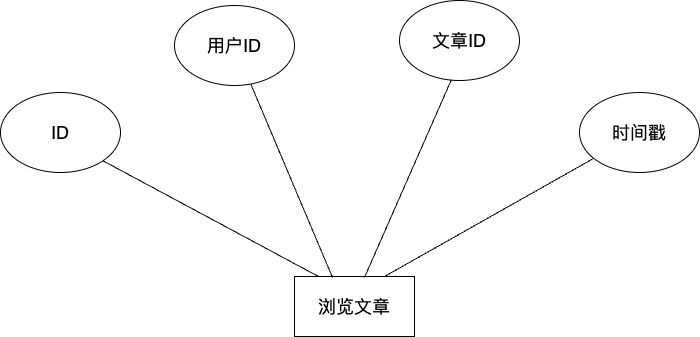


图4.18 浏览文章实体图

Fig.4.18 Browse Article entity chart

（9）浏览时长实体存储了用户ID、类型、浏览时长等信息，如图4.19所示。

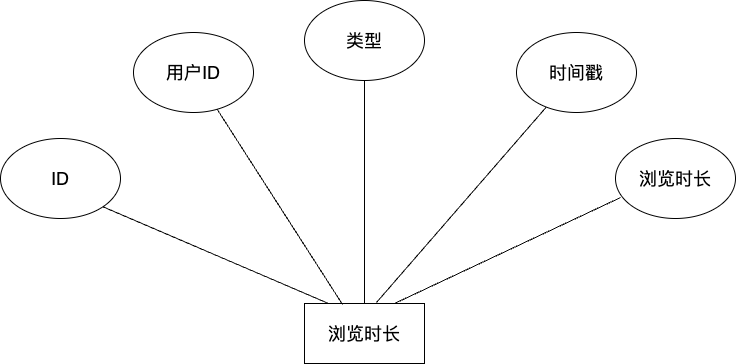


图4.19 浏览时长实体图

Fig.4.19 Browse Duration entity chart

### 4.2.2 数据库逻辑结构设计

（1）用户表有用户ID、用户名、密码、签名等属性，如表4.1所示。

表4.1 用户表

Table 4.1 Users table

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 类型 | 约束 | 描述 |
| id | long | PRIMARY KEY | 用户ID |
| username | varchar | NOT NULL | 用户名 |
| password | varchar | NOT NULL | 密码 |
| age | integer | NOT NULL | 年龄 |
| head\_url | text | NOT NULL | 用户头像 |
| background | text | NOT NULL | 背景图 |
| role | varchar | NOT NULL | 角色 |
| status | varchar | NOT NULL | 状态 |
| birth | long | NOT NULL | 生日 |
| signature | text | NOT NULL | 签名 |
| location | text | NOT NULL | 位置 |
| blog\_address | text | NOT NULL | 博客地址 |

（2）文章表有ID、标题、内容、链接、标签、封面地址等属性，如表4.2所示。

表4.2 文章表

Table 4.2 Articles table

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 类型 | 约束 | 描述 |
| id | long | PRIMARY KEY | 文章ID |
| user\_id | long | Foreign Key | 用户ID |
| username | varchar | Foreign Key | 用户名 |
| author | varchar | NOT NULL | 作者 |
| title | text | NOT NULL | 标题 |
| link | text | NOT NULL | 链接 |
| body | largeText | NOT NULL | 内容 |
| visible\_mode | varchar | NOT NULL | 可见范围 |
| tags | array<text> | NOT NULL | 标签 |
| likes | integer | NOT NULL | 点赞数量 |
| comments | integer | NOT NULL | 收藏数量 |
| views\_num | integer | NOT NULL | 浏览数量 |
| timestamp | long | NOT NULL | 发布时间 |
| cover | text | NOT NULL | 封面 |

（3）收藏表有ID、用户ID、文章等属性，如表4.3所示。

表4.3 收藏表

Table 4.3 Collects table

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 类型 | 约束 | 描述 |
| id | long | PRIMARY KEY | 收藏ID |
| collect\_userId | long | Foreign Key | 用户ID |
| article\_id | long | NOT NULL | 文章ID |
| timestamp | long | NOT NULL | 时间戳 |

（4）评论表有用户ID，文章ID等，如表4.4所示。

表4.4 评论表

Table 4.4 Comments table

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 类型 | 约束 | 描述 |
| id | long | PRIMARY KEY | 评论ID |
| user\_id | long | Foreign Key | 用户ID |
| article\_id | long | Foreign Key | 文章ID |
| content | mediumText | NOT NULL | 评论内容 |
| timestamp | long | NOT NULL | 时间戳 |

（5）朋友表有用户ID，关注用户ID等，如表4.5所示。

表4.5 朋友表

Table 4.5 Friends table

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 类型 | 约束 | 描述 |
| id | long | PRIMARY KEY | 朋友ID |
| user\_id | long | Foreign Key | 用户ID |
| who\_id | long | Foreign Key | 被关注用户ID |
| timestamp | long | NOT NULL | 时间戳 |

（6）点赞表有用户ID，文章ID等，如表4.6所示。

表4.6 点赞表

Table 4.6 Likes table

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 类型 | 约束 | 描述 |
| id | long | PRIMARY KEY | 点赞ID |
| user\_id | long | Foreign Key | 用户ID |
| article\_id | long | Foreign Key | 文章ID |
| timestamp | long | NOT NULL | 时间戳 |

（7）浏览文章表有用户ID，文章ID等，如表4.7所示。

表4.7 浏览文章表

Table 4.7 ViewArticles table

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 类型 | 约束 | 描述 |
| id | long | PRIMARY KEY | 浏览文章ID |
| user\_id | long | Foreign Key | 用户ID |
| article\_id | long | Foreign Key | 文章ID |
| timestamp | long | NOT NULL | 时间戳 |

（8）浏览时长表有用户ID，文章ID等，如表4.8所示。

表4.8 浏览时长表

Table 4.8 ViewDurations table

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 类型 | 约束 | 描述 |
| id | long | PRIMARY KEY | 浏览时长ID |
| user\_id | long | Foreign Key | 用户ID |
| article\_id | long | Foreign Key | 文章ID |
| duration | long | NOT NULL | 浏览时长 |
| timestamp | long | NOT NULL | 时间戳 |

# 第5章 详细系统的实现

详细系统实现细节涉及将系统设计转化为实际运行的软件。这包括编码实践、数据库配置、接口集成、安全策略、错误处理和用户界面开发等。实现细节的准确性直接影响系统的稳定性、性能和用户体验。一个良好的实现应该考虑到代码的可读性和可维护性，以便于团队合作和未来的扩展。

## 5.1 项目整体结构图

### 5.1.1 Android前端

前端应用整体结构分为四个层次，分别是UI层、逻辑层、公共模块层、数据层

UI层由每一个页面所组成，是用户能看到和交互的层面，UI层摈弃了传统的xml形式，使用Jetpack Compose实现，在整个应用中只有一个MainActivity来承载Compose页面。

逻辑层由ViewModel实现，通常一个Compose页面会对应一个ViewModel，它实现UI层和数据层的双向绑定，管理UI状态，解藕UI层和数据层。

公共模块层是逻辑层和数据层都会使用到的，比如数据层Ktor在拦截器中使用Network模块，当没有网络时返回自定义的一个Response，UI层也会监听Network的变化并在无网络或有网络时呈现相应的交互界面。又比如Pager分页模块会在很多有列表的页面或者ViewModel中应用到，实现分页加载。

数据层是应用数据的来源，它包括了本地数据和后台数据。本地数据使用Room数据库存储，它存储了用户的草稿、标签等内容。后台数据是该系统的核心数据来源，通过Ktor Client访问后台数据。

Android前端整体结构图如图5.1所示。

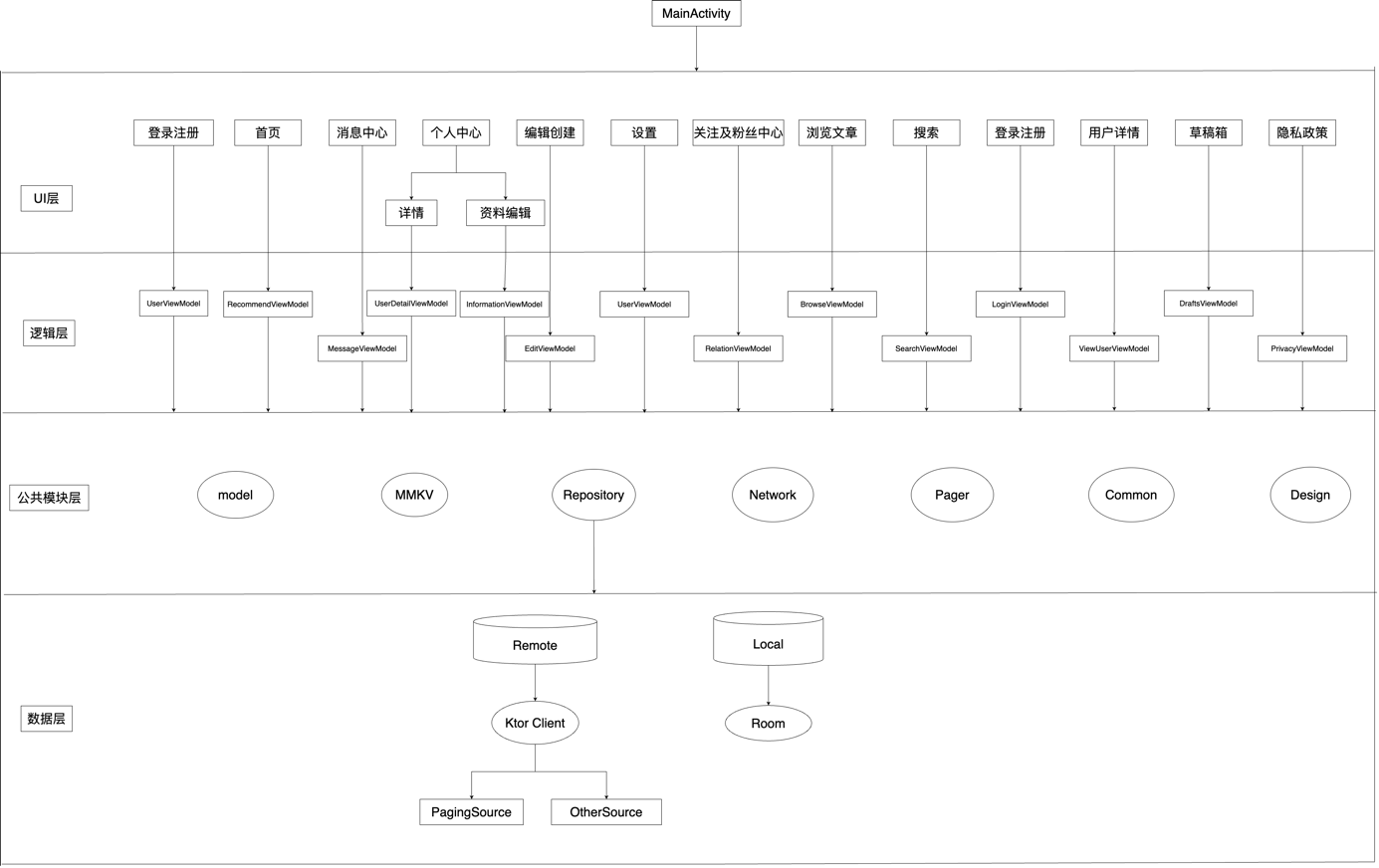


图5.1 Android前端整体结构图

Fig. 5.1 Overall structure of the Android front-end

### 5.1.2 Ktor后台

Ktor后台分为五层，分别是Plugin层、Routing层、公共层、Dao层、数据层。

Plugin层是后台所有业务的核心，它为后台所有业务提供支持。比如Security提供了安全认证和Session能力；Routing提供了网络请求的入口；Databases提供了数据库的配置，比如缓存池、底层使用的数据库MySQL。

Routing层提供了网络接口，比如get、post等，它处理用户发来的请求，并返回相应的数据。

公共层提供了通用能力，比如model库提供了数据模型以及表结构、encrypt提供数据加密及解密能力。

Dao层提供了访问MySQL数据的接口，它使用Exposed实现，每一个操作都是一个事务，保证了数据的一致性和准确性。在Routing层会使用Dao层提供的接口来操纵、处理和返回数据。

数据层服务由MySQL提供，借助于Exposed可以在多种数据库中随意切换而不用改变业务代码，比如当MySQL不再满足本系统要求时，我们可以切换到MongoDB或者其他任意数据库。

Ktor后台整体结构图如图5.2所示。

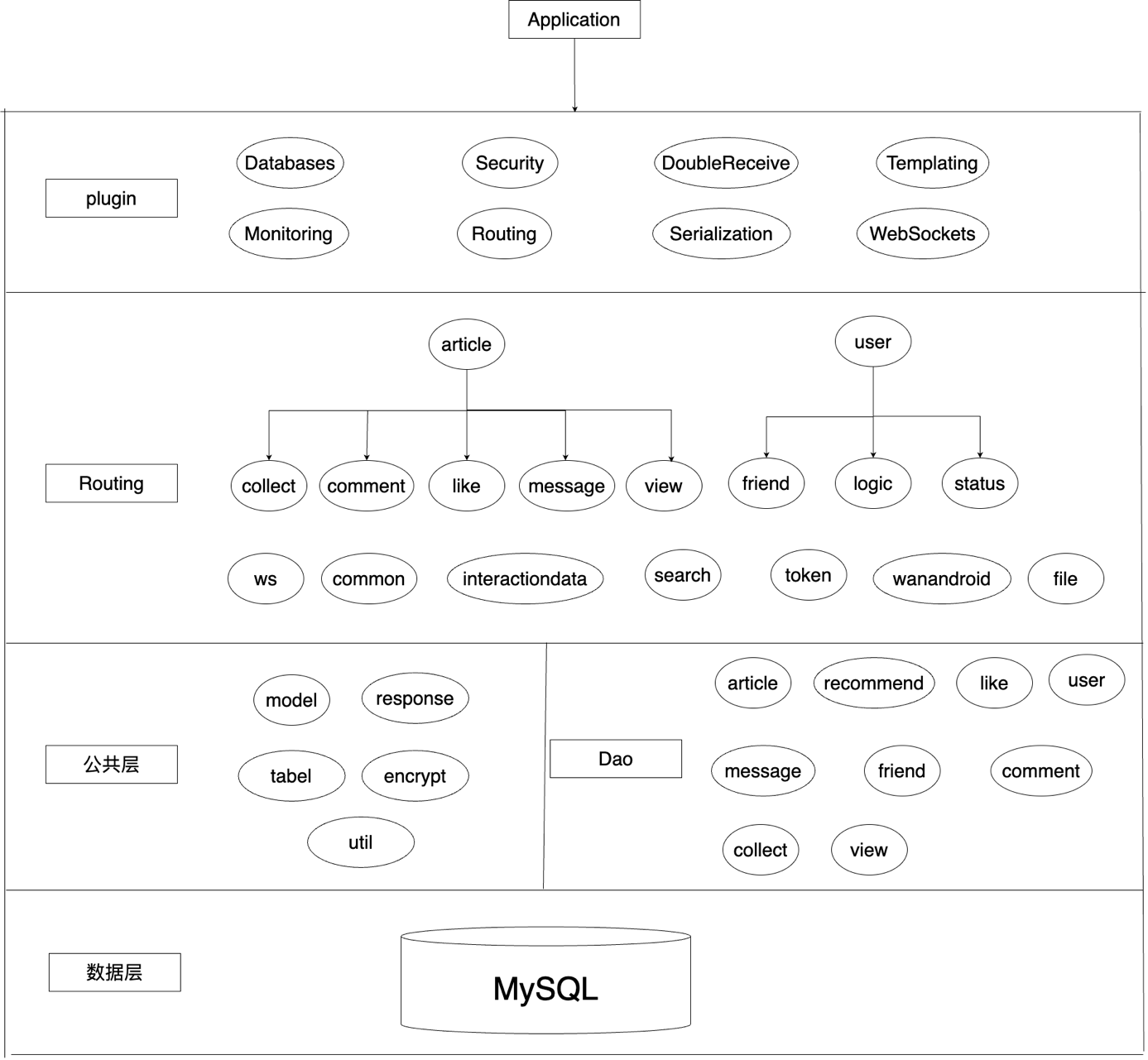


图5.2 Ktor后台整体结构图

Fig. 5.2 Ktor background overall structure diagram

## 5.2 公共层

由于系统中的大多数模块都会使用到公共层的相关能力，所以这里先对公共层进行介绍。

### 5.2.1 存储

在前台，本地存储使用到了MMKV，Room。MMKV是腾讯开源的键值对映射库，可以对少量数据进行高效的存储和查询。Room库是Jetpack Compose中的一个库，它用户存储较大的数据，并且支持数据观察，当数据库中的数据增加、删除或更改时UI层能及时获取到更改并更新UI。

### 5.2.1.1 MMKV

借助于Kotlin代理的便利性，本系统对MMKV做了扩展，可以使用Kotlin的代理方便地操作MMKV。

（1）抽象扩展如下所示：

inline fun <T> delegate(

key: String? = null,

defaultValue: T,

mmapID: String? = null,

crossinline getter: MMKV.(key: String, defaultValue: T) -> T,

crossinline setter: MMKV.(key: String, value: T) -> Boolean

): ReadWriteProperty<Any, T> = object : ReadWriteProperty<Any, T> {

val mmkv = if (mmapID != null) MMKV.mmkvWithID(mmapID) else MMKV.defaultMMKV()

override fun getValue(thisRef: Any, property: KProperty<\*>): T {

return mmkv.getter(key ?: property.name, defaultValue)

}

override fun setValue(thisRef: Any, property: KProperty<\*>, value: T) {

mmkv.setter(key ?: property.name, value)

}

}

（2）有了delegate我们可以方便的实现int、long、bool等类型的存取，例如：

fun long(

key: String? = null,

defaultValue: Long = 0L,

mmapID: String? = null

): ReadWriteProperty<Any, Long> = delegate(

key = key,

defaultValue = defaultValue,

mmapID = mmapID,

getter = { k, v -> decodeLong(k, v) },

setter = { k, v -> encode(k, v) }

)

（3）使用MMKV

object SettingMMKV {

var darkThemeSetting by darkTheme(

key = MMKVKeys.DARK\_THEME\_SETTING\_KEY,

mmapID = currentUser.id.toString()

)

var userHasSetTheme by boolean(

key = MMKVKeys.USER\_HAS\_SET\_THEME\_KEY,

mmapID = currentUser.id.toString()

)

var hasLogout by boolean(

key = MMKVKeys.USER\_HAS\_LOGOUT\_KEY,

defaultValue = true

)

}

### 5.2.2 网络

网络是本系统中的一个重要部分，它支持在无网络时返回自定义Response以及在无网络时展示无网络插画等。

（1）NetworkMonitor，它是监听Android网络状态变化的一个接口，在UI中可以引入该接口的具体实现NetworkMonitorImpl，从而监听网络变化。

interface NetworkMonitor {

val isOnline: Flow<Boolean>

}

class NetworkMonitorImpl(private val context: Context) : NetworkMonitor {

override val isOnline: Flow<Boolean> = callbackFlow {

val connectivityManager = context.getSystemService<ConnectivityManager>()

if (connectivityManager == null) {

channel.trySend(false)

channel.close()

return@callbackFlow

}

val callback = object : NetworkCallback() {

private val networks = mutableSetOf<Network>()

override fun onAvailable(network: Network) {

networks += network

channel.trySend(true)

}

override fun onLost(network: Network) {

networks -= network

channel.trySend(networks.isNotEmpty())

}

}

val request = Builder()

.addCapability(NetworkCapabilities.NET\_CAPABILITY\_INTERNET)

.build()

connectivityManager.registerNetworkCallback(request, callback)

channel.trySend(connectivityManager.isCurrentlyConnected())

awaitClose {

connectivityManager.unregisterNetworkCallback(callback)

}

}.conflate()

}

（2）在Ktor Client中我们查询当前网络状态，如果没网络则返回无网络Response，如下所示：

class NetworkInterceptor : Interceptor {

override fun intercept(chain: Interceptor.Chain): Response {

val request = chain.request()

val app = applicationContext

val connected = app.isCurrentlyConnected()

if (!connected) {

val newRequest = request.newBuilder()

.build()

return buildNetworkErrorResponse(newRequest)

}

return chain.proceed(request)

}

}

### 5.2.3 UI设计系统

UI设计系统决定了本应用中所有页面的组件的颜色、字体等如何显示，是对用户很重要的一部分，它直接影响了用户的使用体验和留存率。该部分主要包括cascade级联菜单、background页面背景色、pager分页库、theme主题、components基础组件库。

1. cascade级联菜单是展示更多操作的一个入口，它通常把多个操作隐藏起来，当用户使用时才弹出。顶层使用接口如下所示：

@Composable

fun <T : Any> CascadeMenu(

modifier: Modifier = Modifier,

isOpen: Boolean,

menu: CascadeMenuItem<T>,

colors: CascadeMenuColors = cascadeMenuColors(),

offset: DpOffset = DpOffset.Zero,

width: Dp = MAX\_WIDTH,

onItemSelected: (T) -> Unit,

onDismiss: () -> Unit,

) {

DropdownMenu(

modifier = modifier

.width(width)

.background(colors.backgroundColor),

expanded = isOpen,

onDismissRequest = onDismiss,

offset = offset

) {

val state by remember { mutableStateOf(CascadeMenuState(menu)) }

AnimatedContent(

targetState = state.currentMenuItem,

transitionSpec = {

if (isNavigatingBack(initialState, targetState)) {

animateToPrevious()

} else {

animateToNext()

}

}

) { targetMenu ->

CascadeMenuContent(

state = state,

targetMenu = targetMenu,

onItemSelected = onItemSelected,

colors = colors,

)

}

}

}

1. background页面背景色被应用于最底部图层，其他UI都显示在该图层之上。主要使用入口如下所示：

@Composable

fun LunimaryGradientBackground(

modifier: Modifier = Modifier,

gradientColors: GradientColors = LocalGradientColors.current,

shape: Shape = RectangleShape,

content: @Composable () -> Unit

) {

val currentTopColor by rememberUpdatedState(gradientColors.top)

val currentBottomColor by rememberUpdatedState(gradientColors.bottom)

GradientBackground(

modifier = modifier,

gradientColors = gradientColors,

shape = shape,

currentBottomColor = currentBottomColor,

currentTopColor = currentTopColor,

content = content

)

}

1. theme主题，theme提供了UI的配色、字体、形状方案，例如深色主题配置：

@VisibleForTesting

val DarkDefaultColorScheme = darkColorScheme(

primary = Pink,

onPrimary = Purple20,

primaryContainer = Purple30,

onPrimaryContainer = Purple90,

...

)

主题实现及使用入口：

@Composable

fun LunimaryTheme(

darkTheme: Boolean = isSystemInDarkTheme(),

dynamicColor: Boolean = false,

content: @Composable () -> Unit

) {

val colorScheme = when {

dynamicColor && supportsDynamicTheming() -> {

val context = LocalContext.current

if (darkTheme) dynamicDarkColorScheme(context) else dynamicLightColorScheme(context)

}

darkTheme -> DarkDefaultColorScheme

else -> LightDefaultColorScheme

}

......

CompositionLocalProvider(

LocalGradientColors provides gradientColors,

LocalBackgroundTheme provides backgroundTheme,

LocaleGreyBackground provides Color.Gray.copy(alpha = 0.7f)

) {

MaterialTheme(

colorScheme = colorScheme,

typography = Typography,

content = content

)

}

}

1. components基础组件库，定义了基本组件，如Button按钮、Tag标签等等，例如LinearButton组件：

@Composable

fun LinearButton(

modifier: Modifier = Modifier,

onClick: () -> Unit,

colors: List<Color> = listOf(

MaterialTheme.colorScheme.primary,

MaterialTheme.colorScheme.secondary

),

textStyle: TextStyle = MaterialTheme.typography.bodyLarge,

height: Dp = 30.dp,

enabled: Boolean = true,

shape: Shape = RoundedCornerShape(16),

text: String

) {

val linearColors = colors.ifEmpty {

listOf(MaterialTheme.colorScheme.primary, MaterialTheme.colorScheme.primary)

}

Surface(

modifier = modifier

.background(

brush = Brush.linearGradient(linearColors),

shape = shape

)

.height(height),

onClick = onClick,

color = Color.Transparent,

enabled = enabled,

shape = shape

) {

Box(contentAlignment = Alignment.Center) {

Text(

text = text,

modifier = Modifier

.padding(horizontal = 8.dp),

color = Color.White,

style = textStyle,

)

}

}

}

### 5.2.4 分页库pager，分页库实现了前端如何向后端分页请求和展示数据，是减轻系统负担的一个重要手段，它可以按需请求和按量返回。

1. pagerFlow是一个通用的方法，分页数据只要实现PageSource，ApppagingSource就会返回特定的数据。请求后端资源接口如下：

fun <T: Any> BaseViewModel.pagerFlow(

sourceFactory: () -> PageSource<T>

) : Flow<PagingData<PageItem<T>>> {

return Pager(defaultPagingConfig) { AppPagingSource(sourceFactory()) }

.flow.cachedIn(viewModelScope)

}

1. ApppagingSource主要方法实现：

override suspend fun load(params: LoadParams<Int>): LoadResult<Int, PageItem<T>> {

return try {

val wishPage = params.key ?: 0

val pageResponse = source.pages(wishPage, DEFAULT\_PER\_PAGE\_COUNT)

if (pageResponse.isSuccess()) {

val data = pageResponse.data

LoadResult.Page(

data = data?.lists?.map {

PageItem(it)

} ?: emptyList(),

prevKey = if (wishPage == 0) null else wishPage - 1,

nextKey = if (wishPage < (data?.pageSize!! - 1)) wishPage + 1 else null

)

} else {

LoadResult.Error(Throwable(message = pageResponse.msg))

}

} catch (e: Exception) {

LoadResult.Error(e)

}

}

### 5.2.5 ViewModel基类实现了网络监听已经paging数据刷新和重试等，主要方法及实现如下：

open class BaseViewModel : ViewModel() {

private val processingMap = ConcurrentSet<String>()

fun fly(url: String, action: () -> Unit) {

if (url !in processingMap) {

processingMap += url

action()

}

}

fun land(url: String) {

if (url in processingMap) {

processingMap.remove(url)

"land $url".logd("fly\_request")

}

}

private val isOnline = networkMonitor.isOnline

.stateIn(

scope = viewModelScope,

started = SharingStarted.WhileSubscribed(5\_000),

initialValue = LunimaryApplication.applicationContext.isCurrentlyConnected(),

)

init { collectNetState() }

private fun collectNetState() {

viewModelScope.launch {

isOnline.collectLatest { nowIsOnline ->

val preOnline = online.value!!

"nowIsOnline=$nowIsOnline, preOnline=$preOnline".logd("App\_is\_off\_line")

if (preOnline) {

if (!nowIsOnline) { // 有网 -> 无网

\_online.postValue(false)

}

} else {

if (nowIsOnline) { // 无网 -> 有网

\_online.postValue(true)

onHaveNetwork()

dispatchOnHaveNetEvent()

}

}

}

}

}

private val onSwitchToHaveNetMap = mutableMapOf<String, () -> Unit>()

fun registerOnHaveNetwork(

key: String,

noNetToHaveNet: () -> Unit

) {

if (onSwitchToHaveNetMap.containsKey(key)) {

return

}

"registerOnHaveNetwork:$key".logd("pagingKey")

onSwitchToHaveNetMap[key] = noNetToHaveNet

}

fun unregisterOnHaveNetwork(key: String) {

if (onSwitchToHaveNetMap.containsKey(key)) {

"unregisterOnHaveNetwork:$key".logd("pagingKey")

onSwitchToHaveNetMap.remove(key)

}

}

fun <T : Any> registerOnHaveNetwork(

keys: List<Pair<String, LazyPagingItems<T>>>

) {

keys.forEach {

val key = it.first

val items = it.second

registerOnHaveNetwork(key) {

items.retry()

}

}

}

fun unregisterOnHaveNetwork(vararg keys: String) {

keys.forEach { unregisterOnHaveNetwork(it) }

}

private fun dispatchOnHaveNetEvent() {

onSwitchToHaveNetMap.values.forEach { it() }

}

fun <T> request(

onSuccess: (data: T?, msg: String?) -> Unit = { \_, \_ -> },

emptySuccess: () -> Unit = {},

onFailed: (msg: String) -> Unit = {},

onFinish: () -> Unit = {},

block: suspend () -> BaseResponse<T>

): Job {

return viewModelScope.launch {

runCatching {

block()

}.onSuccess { response ->

"request success: response=$response".logd()

runCatching {

if (response.isSuccess()) {

onSuccess(response.data, response.msg)

emptySuccess()

response.data ?: run {

"data is null.".logd()

}

} else {

onFailed(response.msg)

}

}.onFailure {

onFailed(it.errorMsg)

}

onFinish()

}.onFailure {

it.message?.loge()

it.printStackTrace()

onFailed(it.errorMsg)

onFinish()

}

}

}

}

### 5.2.6 数据源定义了如何请求后台数据，例如FileSource，它定义了如何上传文件。

interface FileSource {

suspend fun uploadFile(

path: String,

filename: String,

uploadType: Int

): DataResponse<UploadData>

companion object : FileSource by FileSourceImpl()

}

## 5.3 登录注册模块

用户在注册时需要输入合法的账号、密码，合法的用户名不能包含特殊符号，如“+-\*/”等，合法的密码长度必须大于6且只能包含数字、字母和特殊英文字符，注册成功后跳转登录页面。用户登录时需要输入账号和密码进行认证，认证成功后进入平台主页。在登录和注册时如果用户输入的用户米或密码不合法会显示红色的光标作为警示。

（1）在数据层前台登录注册时会调用以下接口中的实现：

interface UserSource {

/\*\* 登录 \*/

suspend fun login(

username: String,

password: String

) : UserResponse

/\*\* 注册 \*/

suspend fun register(

username: String,

password: String

) : UserResponse

}

例如登录login api会构建一个包含用户名和密码的表单并提交，如果验证通过会存储后台返回的session、访问token和刷新token：

override suspend fun login(username: String, password: String): UserResponse {

val response = client.submitForm(

url = loginPath,

formParameters = parameters {

append("username", username)

append("password", password)

}

)

val session = response.headers[MMKVKeys.LUMINARY\_SESSION\_KEY]

?: empty.also { "Session is empty".logd() }

val accessToken = response.headers["access\_token"]

?: empty.also { "Access token is empty".logd() }

val refreshToken = response.headers["refresh\_token"]

?: empty.also { "Refresh token is empty".logd() }

val loginUsername = response.body<UserResponse>().data?.user?.username

if (loginUsername != null) {

"save token&session".logd()

saveSession(loginUsername, session)

saveTokens(TokenInfo(loginUsername, accessToken, refreshToken), loginUsername)

}

return response.body<UserResponse>().init(response)

}

1. 在逻辑层，ViewModel的登录如下所示：

fun login(

username: String,

password: String

) {

request(

block = {

\_loginState.postValue(NetworkResult.Loading())

userSource.login(username, password)

},

onSuccess = { data, \_ ->

if (data?.user != null) {

\_loginState.postValue(NetworkResult.Success(data))

SettingMMKV.hasLogout = false

UserState.updateLocalUser(data.user)

} else {

\_loginState.postValue(NetworkResult.Error(unknownErrorMsg))

}

},

onFailed = {

\_loginState.postValue(NetworkResult.Error(it))

}

)

}

登录注册界面如图5.1所示。

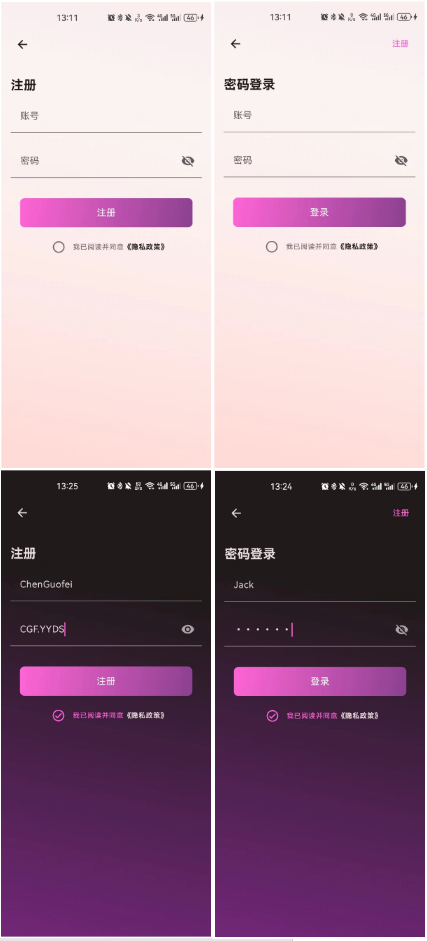


图5.1 用户登录注册界面图

Fig. 5.1 User login and registration interface

## 5.4 首页文章列表模块

首页文章数据主要有三个来源，分别是推荐、全部、朋友，数据获取如以下代码所示：

class RecommendViewModel : BaseViewModel() {

val allArticles: Flow<PagingData<PageItem<Article>>> = pagerFlow { PageAllArticleSource }

val friendsArticles: Flow<PagingData<PageItem<Article>>> = pagerFlow { FriendsPageArticleSource }

val recommendArticles: Flow<PagingData<PageItem<Article>>> = pagerFlow { RecommendArticlesPageSource }

}

首页文章界面如图5.2所示。



图5.2 首页文章列表界面图

Fig. 5.2 Home page article list interface diagram

## 5.5 浏览博客详情模块

浏览博客详情界面如图5.3所示。

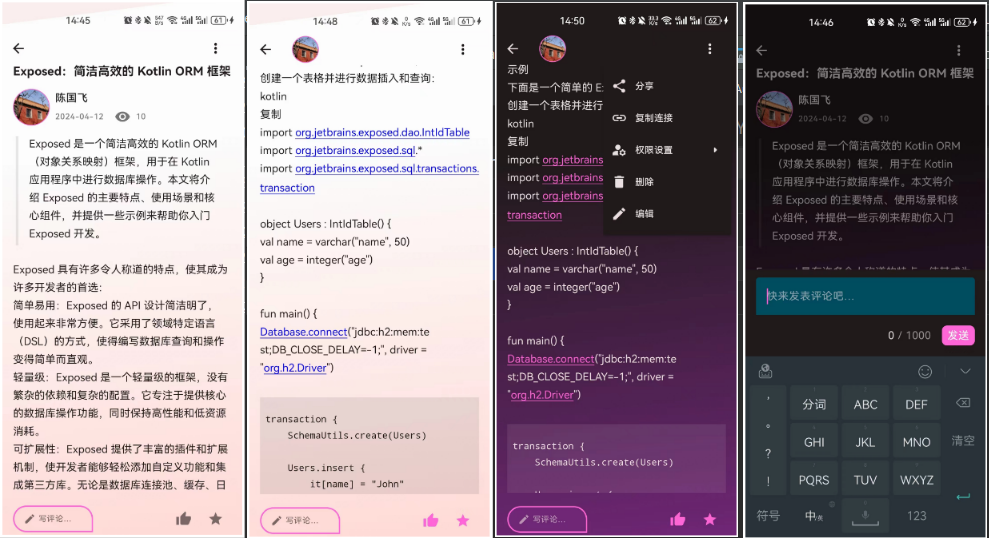


图5.3 博客浏览详情界面图

Fig. 5.3 Browsing Blog Details Interface Image

## 5.5 编辑创建模块

编辑创建界面如图5.4所示。

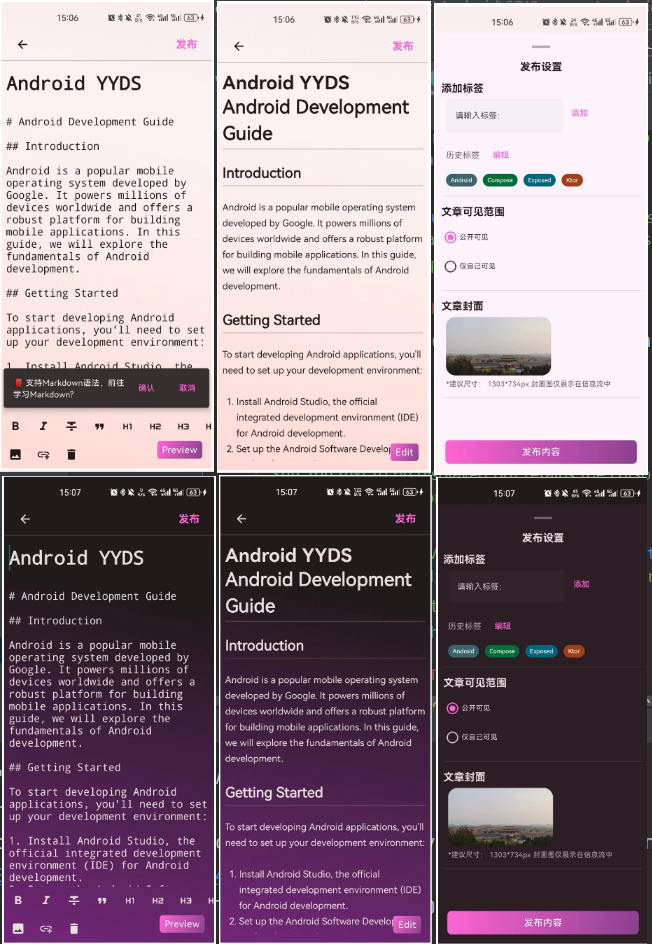


图5.4 编辑创建界面图

Fig. 5.4 Edit and create interface diagram

## 5.6 搜索模块

搜索界面如图5.5所示。

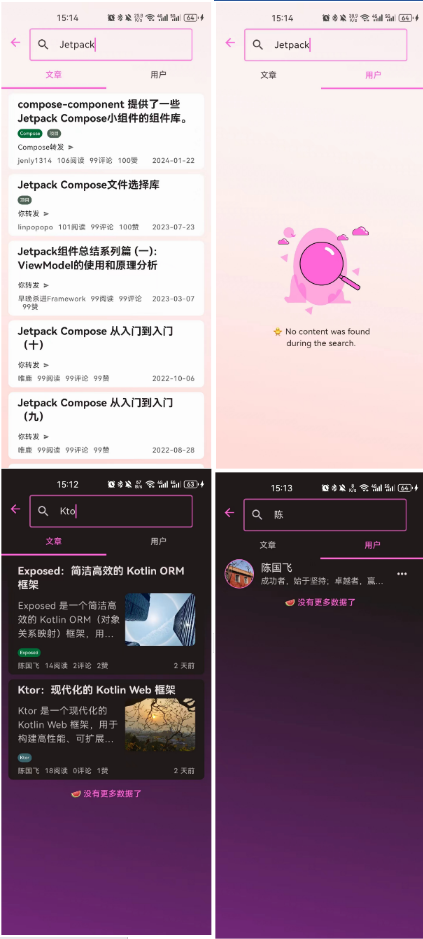


图5.5 搜索界面图

Fig. 5.5 Search interface diagram

## 5.7 消息中心模块

消息中心界面如图5.6所示。



图5.6 消息中心界面图

Fig. 5.6 Message Center Interface Diagram

## 5.8 设置模块

设置界面如图5.7所示。



图5.7 设置界面图

Fig. 5.1 Setting interface diagram

## 5.9 个人中心模块

个人中心界面如图5.8所示。

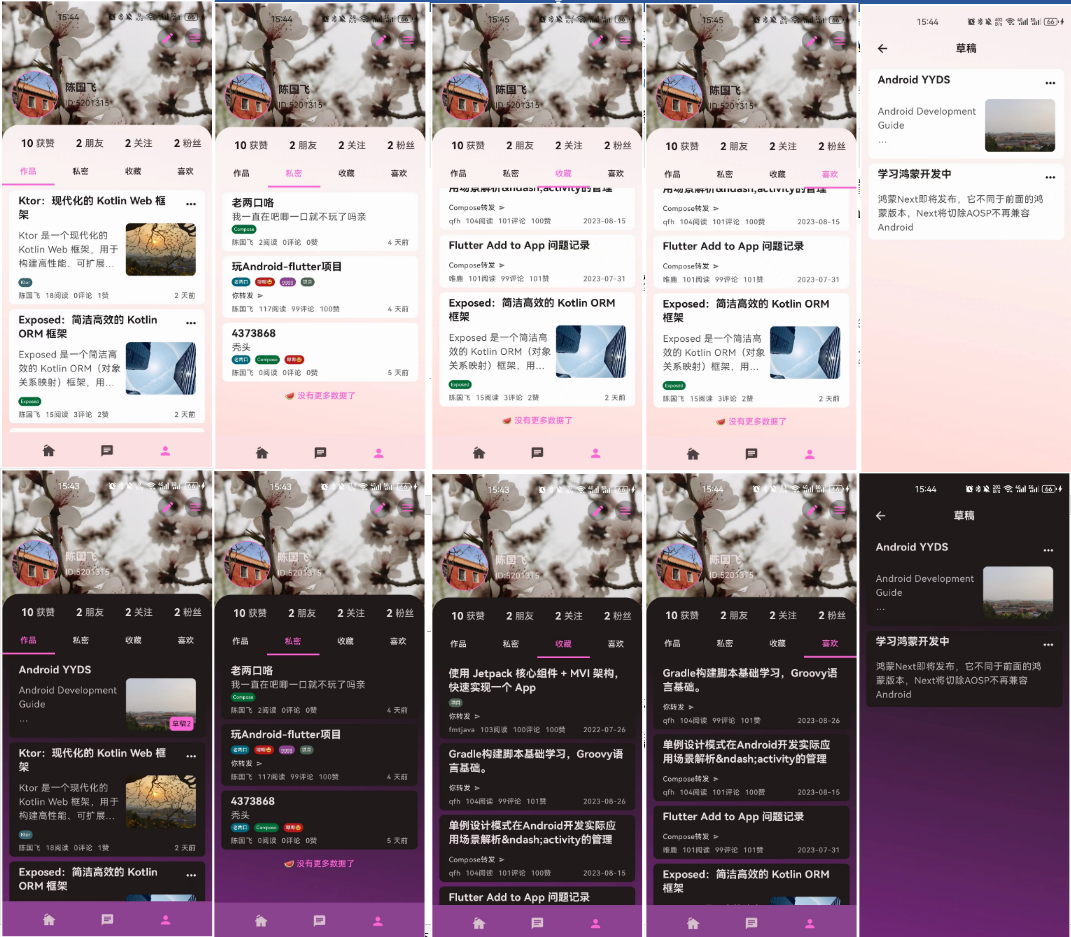


图5.8 个人中心界面图

Fig. 5.8 Personal Center Interface Diagram

## 5.10 粉丝及朋友中心模块

粉丝及朋友中心界面如图5.9所示。



图5.9 粉丝及朋友中心界面图

Fig. 5.9 Fan and Friend Center Interface Diagram

# 第6章 系统测试

系统测试是软件开发生命周期中的一个阶段，旨在验证和评估整个软件系统是否满足设计规格和用户需求。通过执行一系列的测试用例，确保软件的功能、性能、安全性和兼容性等各方面按预期工作。它包括功能测试、性能测试、安全测试等多种测试类型，以检查系统的各个组成部分能否协同工作，以及在实际运行环境中的表现。系统测试的主要作用是发现和修复缺陷、提高软件质量、降低风险，并确保最终交付给用户的产品是可靠和稳定的。

## 6.1 测试方案

本测试方案旨在确保本系统在功能、性能和安全方面达到预定的质量标准。测试将覆盖系统的所有关键组件，包括用户认证、内容管理、界面交互、数据处理和安全性。针对本系统的测试，将分为以下三部分进行测试：

（1）功能测试：包括但不限于用户登录注册、推荐文章列表的加载和现实、收藏、评论、点赞、搜索等功能的正确性。

（2）安全测试：身份验证和会话管理，测试系统防止未经授权访问的机制；数据加密，验证敏感数据的传输和存储是否采用了加密措施；错误处理和信息泄漏，测试系统的错误处理机制以及确保敏感信息不回泄漏。

（3）性能测试：测试应用的响应时间，如冷启动时长、页面加载和交互响应时间。

## 6.2 功能测试

功能测试是软件测试过程中的一项关键活动，它的主要作用是验证软件的各项功能是否符合预先定义的需求。功能测试可以发现与需求不符的行为、设计缺陷和实现错误，在发布之前修正问题，从而提升产品质量，增强用户满意度，并减少维护成本。

### 6.2.1 注册登录功能测试

本系统注册时需满足：

1. 用户名只能由数字、字母、汉字或其他语言文字组成，且长度必须大于等于1，并且不能和已有用户的用户名相同。
2. 密码只能由数字、字母、英文特殊符号组成，且长度必须大于等于8。

测试目的：验证登录和注册功能的正确性、安全性。

测试内容：

1. 输入有效的用户名和密码进行注册，前提：该用户名尚未被注册。
2. 输入有效的用户名和密码进行注册，前提：该用户名已被注册。
3. 输入无效的用户名进行注册。
4. 输入无效的密码进行注册。
5. 输入无效的用户名和密码进行注册。
6. 输入已有用户的用户名和密码进行登录。
7. 输入不存在用户进行登录。
8. 输入错误密码进行登录。

测试结果如表6.1所示。

表6.1登录注册测试

Table 6.1 Login registration test

|  |  |  |
| --- | --- | --- |
| 测试项目 | 测试数据 | 测试结果 |
| （1） | 用户名RadiantDreamer；密码：12345678 | 提示注册成功，跳转到登录页面。 |
| （2） | 用户名ChenGuofei；密码：12345678 | 注册失败，提示“该用户已被注册！” |
| （3） | 用户名Guo\*&+YY；密码：12345678 | 注册失败，提示“用户名只能由数字、字母、汉字或其他语言文字组成” |
| （4） | 用户名ElysianGrace；密码：abc | 注册失败，提示“密码长度应大于或等于8” |
| （5） | 用户名Midnight#\*Glow；密码：a哈d | 注册失败，提示“用户名和密码格式错误” |
| （6） | 用户名ChenGuofei；密码：12345678 | 登录成功，跳转到首页。 |
| （7） | 用户名Object；密码：12345678 | 登录失败，提示“用户不存在” |
| （8） | 用户名ChenGuofei；密码：12345678ops | 登录失败，提示“密码错误” |

### 6.2.2 首页文章列表功能测试

测试目的：

测试内容：

1. 输入有效的用户名和密码进行注册，前提：该用户名尚未被注册。

测试结果如表6.2所示。

表6.2首页文章列表功能测试

Table 6.2 Homepage article list function test

|  |  |  |
| --- | --- | --- |
| 测试项目 | 测试数据 | 测试结果 |
|  |  |  |
|  |  |  |
|  |  |  |

### 6.2.3 浏览博客详情功能测试

测试目的：

测试内容：

1. 输入有效的用户名和密码进行注册，前提：该用户名尚未被注册。

测试结果如表6.2所示。

表6.2首页文章列表功能测试

Table 6.2 Homepage article list function test

|  |  |  |
| --- | --- | --- |
| 测试项目 | 测试数据 | 测试结果 |
|  |  |  |
|  |  |  |
|  |  |  |

### 6.2.4 编辑创建功能测试

测试目的：

测试内容：

1. 输入有效的用户名和密码进行注册，前提：该用户名尚未被注册。

测试结果如表6.2所示。

表6.2首页文章列表功能测试

Table 6.2 Homepage article list function test

|  |  |  |
| --- | --- | --- |
| 测试项目 | 测试数据 | 测试结果 |
|  |  |  |
|  |  |  |
|  |  |  |

### 6.2.5 搜索功能测试

测试目的：

测试内容：

1. 输入有效的用户名和密码进行注册，前提：该用户名尚未被注册。

测试结果如表6.2所示。

表6.2首页文章列表功能测试

Table 6.2 Homepage article list function test

|  |  |  |
| --- | --- | --- |
| 测试项目 | 测试数据 | 测试结果 |
|  |  |  |
|  |  |  |
|  |  |  |

### 6.2.6 消息中心功能测试

测试目的：

测试内容：

1. 输入有效的用户名和密码进行注册，前提：该用户名尚未被注册。

测试结果如表6.2所示。

表6.2首页文章列表功能测试

Table 6.2 Homepage article list function test

|  |  |  |
| --- | --- | --- |
| 测试项目 | 测试数据 | 测试结果 |
|  |  |  |
|  |  |  |
|  |  |  |

### 6.2.7 设置功能测试

测试目的：

测试内容：

1. 输入有效的用户名和密码进行注册，前提：该用户名尚未被注册。

测试结果如表6.2所示。

表6.2首页文章列表功能测试

Table 6.2 Homepage article list function test

|  |  |  |
| --- | --- | --- |
| 测试项目 | 测试数据 | 测试结果 |
|  |  |  |
|  |  |  |
|  |  |  |

### 6.2.8 个人中心功能测试

测试目的：

测试内容：

1. 输入有效的用户名和密码进行注册，前提：该用户名尚未被注册。

测试结果如表6.2所示。

表6.2首页文章列表功能测试

Table 6.2 Homepage article list function test

|  |  |  |
| --- | --- | --- |
| 测试项目 | 测试数据 | 测试结果 |
|  |  |  |
|  |  |  |
|  |  |  |

### 6.2.9粉丝及朋友中心功能测试

测试目的：

测试内容：

1. 输入有效的用户名和密码进行注册，前提：该用户名尚未被注册。

测试结果如表6.2所示。

表6.2首页文章列表功能测试

Table 6.2 Homepage article list function test

|  |  |  |
| --- | --- | --- |
| 测试项目 | 测试数据 | 测试结果 |
|  |  |  |
|  |  |  |
|  |  |  |

## 6.3 安全测试

安全测试是系统测试中的一个重要方面，是评估软件系统抵御恶意攻击和未授权访问能力的过程。它的作用是发现和修复软件中的安全漏洞，包括代码层面的弱点、配置错误、不安全的数据存储和传输方式。

### 6.4.1 认证和授权测试

### 6.4.2 ……测试

## 6.4 性能测试

性能测试的作用是确保应用在各种条件和设备上都能提供流畅、响应迅速的用户体验。它涉及评估应用的响应时间、内存使用、CPU 耗电量、网络请求效率等关键性能指标。通过模拟不同用户场景，性能测试确保应用在各种设备配置和操作条件下都能保持稳定运行，从而提升用户满意度，降低应用崩溃率和提高用户留存率。

### 6.4.1 冷启动时长

# 第7章 总结与展望

## 7.1 总结

本论文全面阐述了 LunimaryBlogs 博客系统的设计与实现，涵盖了系统架构、功能设计、技术选型、开发过程和测试策略。通过采用 Kotlin 作为主要开发语言，结合 Ktor Server 和 Exposed 库构建后端服务，以及使用 Jetpack Compose 和 Ktor Client 开发前端用户界面，本系统成功实现了一个现代化、响应式和用户友好的博客平台。

系统的核心功能，包括用户注册与登录、博客创作与浏览、评论与点赞、以及个性化内容推荐等，均已按照预期工作，为用户提供了丰富的交互体验。性能测试结果表明，LunimaryBlogs 在处理高并发请求和数据密集型操作方面表现出色，同时安全测试也确保了系统的数据完整性和用户隐私安全。

未来工作将侧重于平台的功能拓展、性能优化和跨平台兼容性，以及引入人工智能技术以增强个性化用户体验。LunimaryBlogs 博客系统的开发实践不仅展示了 Kotlin 全栈开发的潜力，也为构建高效、安全且易于维护的博客系统提供了宝贵的经验。随着技术的不断进步和用户需求的演变，LunimaryBlogs 期待在未来的发展中继续提供创新和价值。

## 7.2 未来工作展望

在未来的工作中，本系统将继续追求技术创新和功能完善。随着Kotlin生态的不断成熟和发展，将计划利用Kotlin Multiplatform的能力，探索共享代码库以实现跨平台功能，从而扩展到iOS等其他平台，提供更加统一的用户体验。同时，将继续关注Jetpack Compose for Desktop和Web的进展，以期将该系统前端应用扩展到桌面和Web环境。

在后端，将继续优化Ktor Server的性能，探索异步编程和协程的更多可能性，以提高并发处理能力和响应效率。对于数据库操作，将进一步优化Exposed框架的使用，提升数据处理的安全性和效率。

安全性作为系统的重中之重，未来将持续加强安全机制，包括但不限于增强数据加密、完善认证授权流程、防御网络攻击等方面。此外，还计划引入更多的智能化功能，如利用机器学习算法对用户行为进行分析，从而提供更为精准的内容推荐和个性化服务。

最后，我会密切关注用户反馈和市场动态，不断迭代产品，增加新的功能模块，提升系统的可用性和可维护性。通过持续的技术创新和用户体验优化，本系统致力于成为博客领域的佼佼者。

# **参考文献:**

[1].实体店生死局[J].21世纪商业评论,2022(06):18-19.

[2]周野,李蓝.一种低代码开发思路,解惑中小企业智能改造困局[J].智能制造,2023(01):116-120.

……

[10] Lemire, D., & Keiser, J. (2023). On-Demand JSON: A Better Way to Parse Documents?.

[11]Amaro Gonçalo,Moutinho Filipe,CamposRebelo Rogério,Köpke Julius & Maló Pedro.(2021).JSON Schemas with Semantic Annotations Supporting Data Translation. Applied Sciences(24). doi:10.3390/APP112411978.

# **致谢:**

在 LunimaryBlogs 博客系统项目的研究与开发过程中，我获得了来自多方面的宝贵支持与帮助，在此我要表达我最诚挚的感谢。

首先，我要感谢我的导师，他们不仅在技术上给予我指导，还在研究方法和论文撰写上提供了宝贵的建议。他们严谨的学术态度和对细节的关注激励我不断前进。

我还要感谢参与测试的用户，他们提供了反馈和建议，帮助我改进系统，并使其更加贴合用户需求。

此外，我要感谢提供技术支持和软件工具的开源社区，特别是 Kotlin、Ktor 和 Jetpack Compose 的开发团队，他们的辛勤工作为我们的项目提供了坚实的基础。另外，我还要感谢Techflow，一位bilibili博主，是他的视频让我选择了Ktor作为后端开发的技术栈，并在之后的开发中给我带来了很大的便利。

最后，我要感谢我的家人对我的支持和鼓励，他们为我提供了一个安静的研究环境，让我能够专注于我的项目。