

華中科技大学

《软件工程》项目报告

题目：

院 系 _____ 计算机学院 _____

专业班级 _____ 大数据 2101 班 大数据 2102 班 _____

指导教师 _____ 潘鹏 _____

组 名 _____ OmegaXYZ _____

姓 名 _____ 张钧玮 马耀辉 夏彦文 _____

学 号 _____ U202115520 U202115638 U202115317 _____

2023 年 11 月 30 日

目 录

1	任务书	1
2	问题定义	2
2.1	项目背景与意义	2
2.2	项目基本目标	3
2.3	可行性分析	3
2.4	人员管理和项目进度管理	5
3	需求分析	8
3.1	需求分析概述	8
3.2	UML 相关需求分析图	8
3.3	数据流分析图	10
3.4	系统 ER 图	11
3.5	原型系统设计	13
4	概要设计和详细设计	16
4.1	后端	16
4.2	前端	41
5	实现与测试	46
5.1	前后端通信	46
5.2	前端路由	54
5.3	前端页面显示	56
5.4	结果分析	58
6	总结	59
6.1	用户反馈	59
6.2	全文总结	59
7	体会	60
8	附录	61

1 任务书

1.1.1 总体要求

1. 综合运用软件工程的思想，协同完成一个软件项目的开发，掌软件工程相关的技术与方法；
2. 组成小组进行选题，通过调研完成项目的需求分析，并详细说明小组成员的分工、项目的时间管理等方面。
3. 根据需求分析进行总体设计、详细设计、编码与测试等。

1.1.2 基本内容

1. 问题概述、需求分析：正确使用相关工具和方法说明所开发软件的问题定义和需求分析，比如 NABCD 模型，Microsoft Visio，StarUML 等工具 (20%)；
2. 原型系统设计、概要设计、详细设计：主要说明所开发软件的架构、数据结构及主要算法设计，比如墨刀等工具 (35%)；
3. 编码与测试：编码规范，运用码云等平台进行版本管理，设计测试计划和测试用例 (30%)；
4. 功能创新：与众不同、特别吸引用户的创新 (10%)；
5. 用户反馈：包括用户的使用记录，照片，视频等 (5%)。

2 问题定义

2.1 项目背景与意义

2.1.1 项目背景

“众人拾柴火焰高”，集思广益，有助于提高群体的效率；“兼听则明，偏听则暗”，他人的意见也会带给我们新的启迪和思考。随着社会不断发展，人们在社会生活中的议论和参与变得更加丰富多样。

进入信息时代，互联网论坛成为人们交流意见的新平台。近年来，前台匿名聊天论坛的兴起为网络交流注入新的活力。目前，许多高校学生主动创建了匿名聊天论坛，例如“树洞”，然而，这些树洞也暴露出一系列值得关注的问题。例如，管理员难以有效监管可能出现的违法违规内容；系统开发不够完善，存在用户数据泄露的潜在风险；在高流量场景下，服务可能崩溃，而且论坛形式相对单一等。

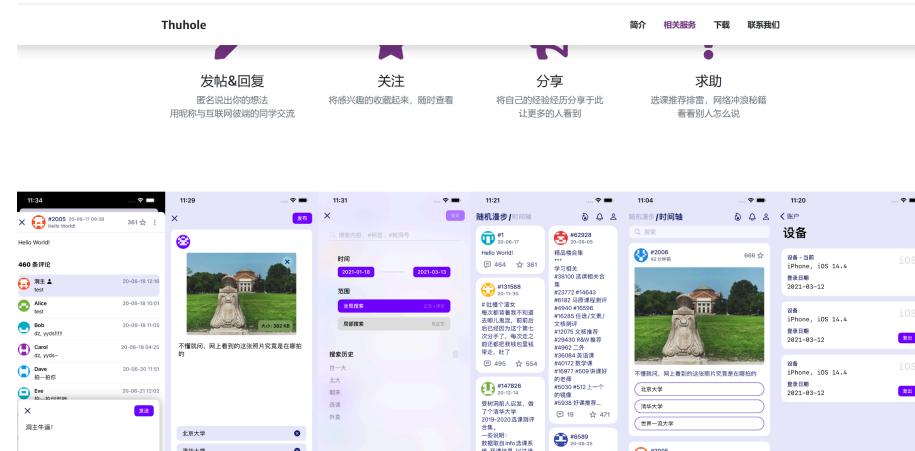


图 1：“国内某大学树洞网页”

由此可见，目前我们高校“树洞”需要的软件应满足以下的需求：易于管理员管理；数据安全性、可靠性高；提供比较完善的匿名机制，让同学们在不违法违规的前提下畅所欲言。

2.1.2 项目意义

项目的意义在于迎合背景中提到的需求、解决部分现存问题，并优化高校匿名聊天论坛的体验。首先，创建一个易于管理员管理的平台可以更有效地监督内容，确保论坛的讨论既自由又合规。其次，提升数据安全性和可靠性至关重要，以保护用户的隐私和避免数据泄露。此外，完善的匿名机制可以鼓励更多学生参与讨论，

分享他们的想法和经验，而不必担心个人信息泄露或遭受偏见。这样，论坛不仅能够成为一个表达和交流的空间，也能成为学习和成长的平台。

通过这个项目，我们不仅可以提升高校社区的交流质量，还能促进知识和经验的共享，帮助学生们在学术和个人成长方面得到更好的支持。此外，改进的论坛还可以成为处理紧急情况和提供心理健康支持的重要资源。总的来说，这个项目的意义在于创造一个更安全、更包容、更高效的线上交流环境。

2.2 项目基本目标

通过深入研究现实论坛中帖子的发布、浏览、评论及删除等环节，我们计划开发一款基于 Tailwind css、Vuejs、Spring boot 及 Mysql 技术的前后端分离 Web 应用。该应用不仅旨在便利论坛管理员对各类帖子信息的有效管理，而且着眼于为用户提供便捷、安全的帖子获取体验，保障个人信息的安全，并支持自由发表评论。我们的目标是为同学们提供一个既适合休闲上网，又能进行学术交流和生活分享的平台，同时配备严密的隐私保护机制，以确保一个安全、包容、互动丰富的在线社区环境。

2.3 可行性分析

通过以上对项目背景、项目意义和项目基本目标的考察研究，并结合自身能力、项目开发平台等因素进行考量，最终可以从以下几个方面分析项目的可行性：

2.3.1 技术难度

本次项目基于前后端分离进行 Web APP 的开发工作，前端主要使用 Tailwind Css 构建网页布局，用 JS + Vue 框架编写页面变换逻辑，后端主要使用 Spring Boot 框架，使用 Spring MVC 进行 Web 开发，MySQL 作为数据库，MyBatis-Plus 作为数据访问框架，并基于 Java 17。数据以 JSON 形式在前端和后端之间传送。负责前端的同学和负责后端的同学都有一定程度的相关技术基础。

在图形界面方面，Tailwind CSS 为开发者提供了丰富的系统控件，使得我们可以很轻松的编写出漂亮的界面。在数据存储方面，我们选用 Mysql 这种经典的关系数据库存储数据。它不仅支持标准的 SQL 语法，还提供了 java/js API 进行操作，让存储和读取数据变得非常方便。

综上，系统在技术方面是可行的。



图 2: 项目技术栈: Vue, Tailwind Css, Spring Boot...

2.3.2 市场需求

根据市场情况、口头调查结果和 NABCD 模型分析可以得出，“树洞”论坛有着比较广阔的应用场景，很受青年学生欢迎。从用户需求上看，很多同学希望能有一个匿名的、氛围友好的网上论坛，用于课余时间的交流和讨论。

从市场竞争上看，目前大部分“树洞”和网上匿名论坛有提供的服务不够完善、部分用户发言不友善，导致管理困难、用户隐私信息安全性不高等，因此我们的项目有很大的竞争力。在产品推广上，我们可以借助校内自媒体传播、在学生宿舍门口张贴海报宣传等，还可以和校心理协会等组织合作，帮助情绪上处于低谷的同学走出困境。

 中国政府网
https://www.gov.cn/xinwen/content_2678868 :

争议中的另类APP“秘密”背后的秘密_中华人民共和国中央 ...

2014年5月13日 — 专家指出，这种匿名性提供一个人人皆可倾诉烦恼和压力的“树洞”，而且还有大量的匿名评论，用户之间可以互相交流，尤其是那些有着相同经历者，答疑解惑互相 ...

 央视网
<http://news.cntv.cn> 新闻台, 新闻中心 :

树洞微博在争议声中走红勿成泄愤人身攻击平台

树洞微博在争议声中走红勿成泄愤人身攻击平台，“树洞其实就是个微博代理发布端”，华中科技大学曾参与树洞微博开发的小汪说。针对树洞微博里不少或抱怨或“晒”隐私的内容 ...

图 3: “‘树洞’引发的社会关注”

2.3.3 开发成本

1. 人力成本 本次项目工作量稍大，因而有一定的开发成本，但考虑到开发小组有三名成员，部分成员有 Web 开发技术基础，因而人力成本在可以接受的范围内。在时间成本上，虽然开发设计的部分技术同学们前期学习了解不多，但网络上可以搜到丰富的学习资料等，因而初期学习所需要的时间成本也不会很多，小组成员每周进行几个到十几个小时的代码工作最终可以完成开发任务。

2.经济成本：本项目计划采用开源框架和开源工具，经济成本花费很小；对于可能需要付费的 visio 等绘图软件，利用学校提供的正版软件许可，不需要额外的开销。后期如果要部署到有公网 IP 的服务器上，可能需要向学校申请校园网公网 IP，或者通过云服务器提供服务，也仅需要很少的运维花费。此外，本次软件开发中，编写、测试代码的硬件设备使用同学们自己的个人笔记本，不需要购置额外的硬件设备。从这些观察中可以看出：在本项目的整个周期内，花费的经济成本非常小。

2.4 人员管理和项目进度管理

2.4.1 人员分工

我们首先对项目的总体任务进行分工。我们要实现的是一个前后端分离的“树洞”系统，所以任务首先可以分为前端和后端两个部分。综合考虑小组成员技术积累、时间投入和技术兴趣因素，我们初步决定：马耀辉同学负责后端任务，夏彦文和张钧炜同学负责前端任务。前端任务继续细分成两部分：网页代码部分和前端代码部分。这两个部分还可以再做拆解，如下图所示。

如下图所示，任务共可以被拆分成几个阶段。每个阶段内的前后端任务可以并行，从而充分提高开发效率。

2.4.2 源代码管理

首先，我们在 Github 平台上建立了一个组织，在这个组织下面建立了仓库，如下图所示。

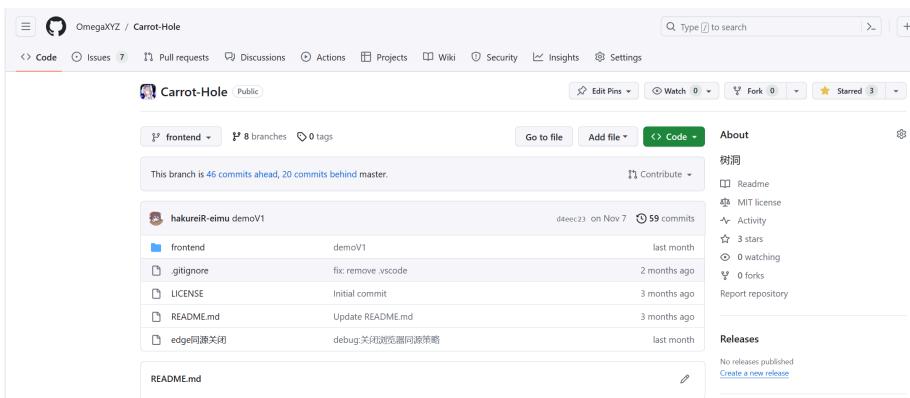


图 4: “Github 平台上的组织和仓库”

在仓库中，我们根据个人分工的不同创建了多个不同的分支，分支命名规范为“人名缩写/分工”。我们在不同的分支上开发，完成到一定程度时合并分支到该分工的开

发分支，最终合并到 master 分支。对于 Commit 信息，我们也严格遵照规范，增强了可读性，如下图所示。

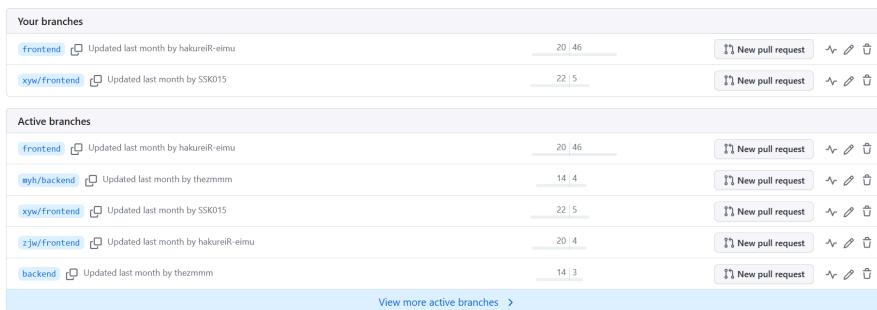


图 5: “各项任务的开发分支”

feat: change sidebar SSK015 committed on Nov 4	2c1f493
test: testData and make FullPoster Box extendible SSK015 committed on Nov 4	863cd55
debug:关闭浏览器同源策略 hakureiR-eimu committed on Nov 4	71489ac
bug:cros错误 hakureiR-eimu committed on Nov 4	6379d60
feat: Add Review Box SSK015 committed on Nov 4	a3ce6e3
feat: merge request SSK015 committed on Nov 4	4e69e08

图 6: “规范提交记录”

2.4.3 进度管理

在项目开发前期我们每隔四五天左右开一次小组会议，确定了设计目标，分析了可行性；项目开发中期、后期，我们在群聊里及时共享进度、互相询问技术问题，保证了进度良好、有序推进。

任务编码	任务名称	开始时间	结束时间	工期(天)	进度(%)
1 ● /	□ 开发	■ 2023-09-24	■ 2023-11-07	45	100.00
2 ● 1	前端开发	■ 2023-09-24	■ 2023-10-25	32	100
3 ● 2	初始化和环境搭建	■ 2023-09-24	■ 2023-09-30	7	100
4 ● 3	界面设计	■ 2023-10-06	■ 2023-10-11	6	100
5 ● 4	功能实现（发帖、评论、权限）	■ 2023-10-12	■ 2023-10-19	8	100
6 ● 5	测试	■ 2023-10-20	■ 2023-10-25	6	100
7 ● 6	后端开发	■ 2023-09-24	■ 2023-10-25	32	100
8 ● 7	环境搭建和数据库准备	■ 2023-09-24	■ 2023-09-30	7	100
9 ● 8	API设计和开发	■ 2023-10-06	■ 2023-10-19	14	100
10 ● 9	安全性和权限控制	■ 2023-10-20	■ 2023-10-22	3	100
11 ● 10	测试	■ 2023-10-23	■ 2023-10-24	2	100
12 ● 11	集成和部署	■ 2023-10-26	■ 2023-11-07	13	100
13 ● 12	前后端集成	■ 2023-10-26	■ 2023-10-30	5	100
14 ● 13	系统测试	■ 2023-10-31	■ 2023-11-04	5	100
15 ● 14	用户文档撰写	■ 2023-11-05	■ 2023-11-07	3	100

图 7: “工期安排”

我们在分支提交信息里明确写出进度，小组成员先查看 Github 上的提交记录，就清楚了大部分任务的进度情况。

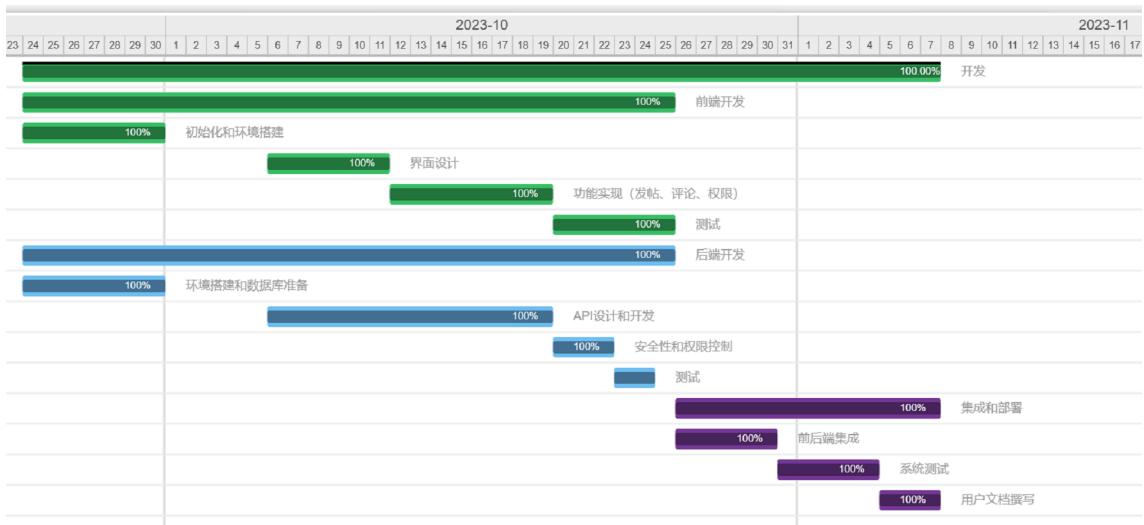


图 8: “进度管理-甘特图”

3 需求分析

3.1 需求分析概述

- N (需求)

在日常生活中，学生面临着各式各样的压力和挑战，往往缺乏合适的倾诉对象或在某些情况下不便于实名分享自己的感受和想法。这种背景下，建立一个能够提供匿名交流的在线平台变得尤为重要。我们计划开发的“树洞”平台，旨在满足这一需求，提供一个安全、匿名的环境，使同学们能够在放松和支持的氛围中交流，分享日常生活中的压力和经历，同时也强化了同学间的联系。

- A (方法)

该平台将包含用户登录、发帖、回帖等基本功能。所有用户的真实信息将被安全地保存在后端数据库中，而用户在前台的活动如发帖和回帖将保持匿名。这种设计不仅能够根据用户的权限级别展示不同的帖子，还能提供诸如发帖、回帖、删除帖子等功能，确保用户在享受交流自由的同时，其个人信息得到妥善保护。

- B (好处)

此平台的主要好处在于为同学们提供了一个安全、自由的交流空间。通过前台匿名的方式，用户可以自由地发帖和回帖，同时平台能够有效监控并排查发布违法或违规内容的账号信息。此外，通过用户权限的隔离，进一步保证了用户个人隐私的安全。这样的设计也使得平台更易于管理和维护。

- C (竞争)

与其他类似的“树洞”软件相比，我们的平台提供了更自由、更即时、更轻量化的内容访问和发布功能。许多现有的树洞平台在处理违法或违规信息方面存在不足，而我们的系统通过后端数据库的维护，能够更有效地进行删帖、封号等管理活动，从而提供一个更安全、更负责任的交流环境。

- D (推广)

为了推广我们的平台，我们计划在学校的交流群组中进行宣传，并与院系学生组织合作，通过这些组织的网络和资源进行有效推广。通过这种方式，我们能够直接接触到潜在的用户群体，从而提高平台的知名度和使用率。

3.2 UML 相关需求分析图

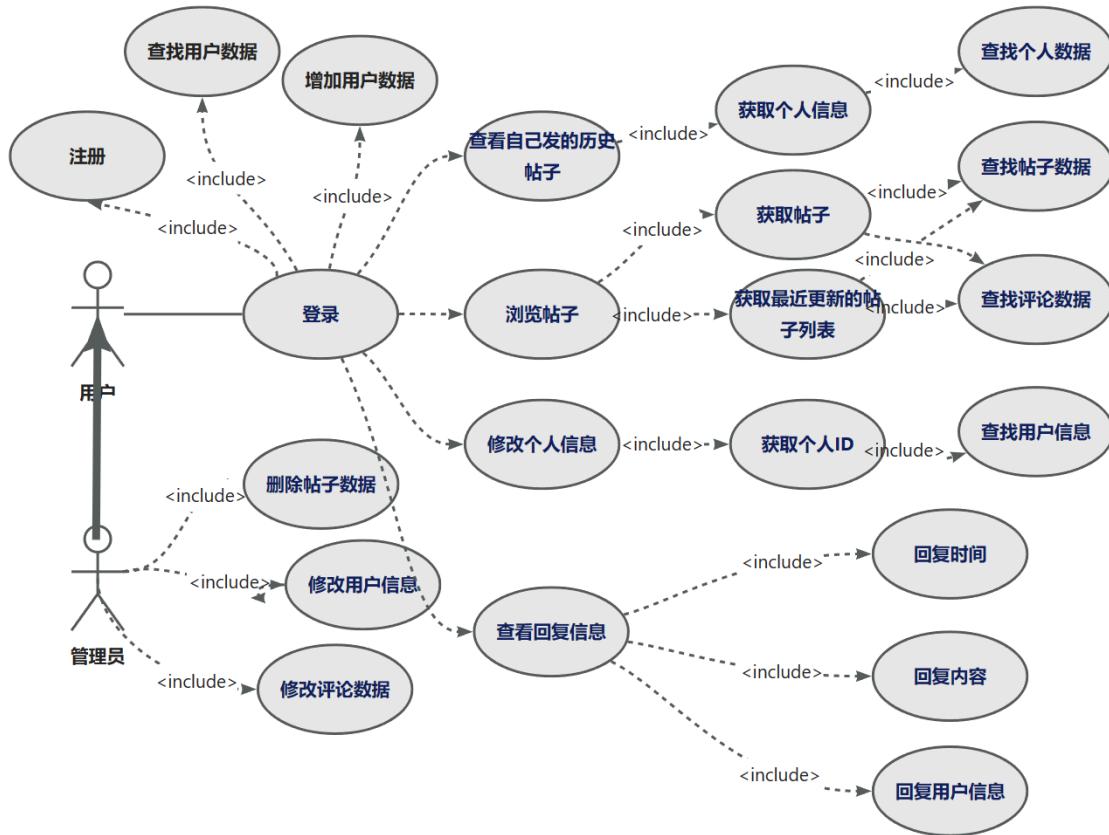


图 9: “UML 需求分析”

我们系统的 UML 需求分析图如上图所示。在这张图中，我们看到了两类主要的参与者：普通用户和管理员，以及他们可以执行的操作。

该用例图详细描述了树洞论坛系统中用户和管理员的交互活动。普通用户的活动包括注册和登录，这是访问论坛的先决条件。一旦进入系统，用户可以浏览帖子来获取信息，或使用搜索功能来定位特定的内容。用户互动的核心在于发帖和回帖，这些操作为社区的沟通提供了基础。此外，用户还可以编辑或删除自己的帖子，这样的功能保障了他们可以控制自己在论坛上的信息和隐私。

管理员在登录后获得的权限更为广泛，涵盖了帖子的全面管理。他们不仅可以浏览所有帖子以监督论坛活动，还能进行帖子的审核和删除，确保论坛内容的质量和适宜性。管理活动还包括用户账户的管理，管理员可以对用户的活动进行必要的监控和制裁。

在图中，用虚线和“include”标签表示的关系表明某些活动是其他活动的必要部分。例如，发帖通常会“include”编辑帖子的功能。整体上，这个用例图提供了一个

清晰的视图，描绘了树洞论坛系统中的角色和他们可以执行的操作，以及这些操作之间的依赖关系。

3.3 数据流分析图

在系统中，用户和管理员是主要的参与者，他们与系统的不同功能进行交互。

下面是对下图展示的每部分的详细解释：

- **用户（User）：**

用户是系统的主要操作者之一。他们可以进行多种操作，如发布帖子、浏览帖子、评论、个人信息编辑、查看通知等。

- **管理员（Admin）：**

管理员是具有特殊权限的操作者。他们负责管理帖子，包括删除帖子和审核帖子内容，以确保内容的合理性和适宜性。

- **系统功能：**

发布帖子：用户创建并发布新的讨论帖。 浏览帖子：用户和管理员查看论坛上的帖子。 评论：用户对帖子添加评论。 个人信息编辑：用户更新他们的个人账户信息，如密码、电子邮件地址等。 查看通知：用户接收系统消息和帖子更新通知。 帖子管理：管理员对帖子进行审核和管理。

- **数据存储：**

用户数据库：存储用户个人信息的数据库。 帖子数据库：存储帖子和评论的数据库。

- **数据流：**

数据流向从用户和管理员指向不同的系统功能，表示信息的输入或操作的请求。反向的数据流向表示系统对操作的响应，如返回浏览结果、通知信息、管理结果等。

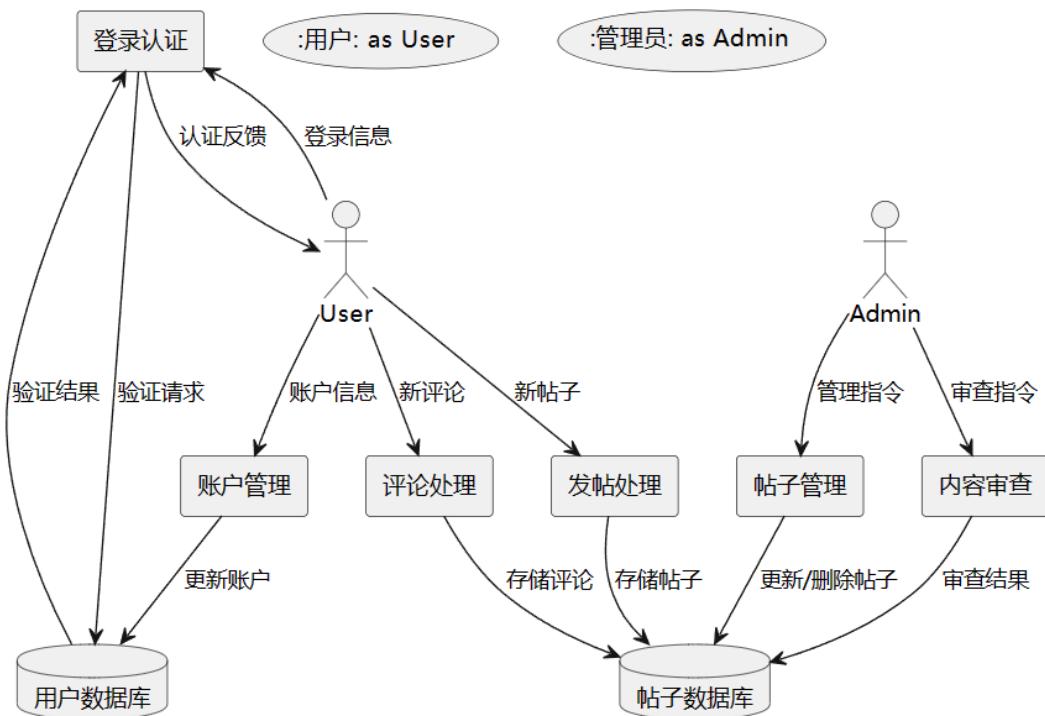


图 10: “数据流分析图”

3.4 系统 ER 图

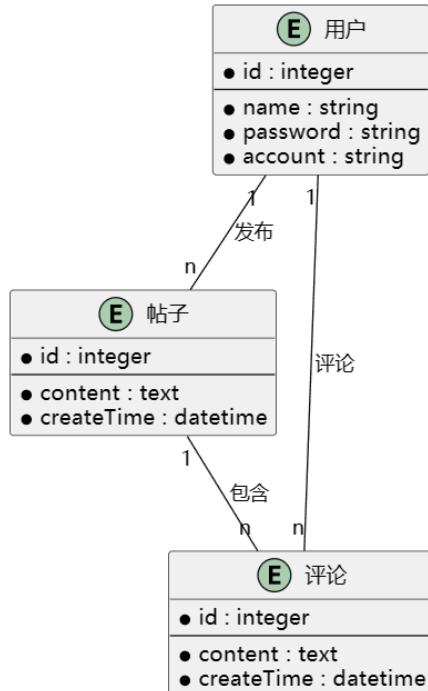


图 11: “系统 ER 图”

ER 图主要包含三个实体：用户、帖子和评论。每个实体都有其唯一的属性和标识符。

用户实体代表论坛的注册成员，它包含了用户的基本信息。每个用户都有一个唯一的 ID，这是他们在系统中的唯一标识。除此之外，用户实体还包含姓名、密码和账号信息。姓名用于标识用户的真实或昵称，密码用于登录验证（在实际存储时需要加密），而账号可能是用户的电子邮件地址或学号，用于登录和用户识别。

帖子实体代表用户在论坛上发布的内容。每个帖子都有一个唯一的 ID，以及包含其详细信息的文本内容和创建时间戳。帖子的内容是用户交流和分享信息的主体，创建时间则记录了帖子发布的具体时间。

评论实体是对帖子的回应。与帖子类似，每个评论都有一个唯一的 ID，内容和创建时间。评论允许用户对帖子内容进行讨论和回复，增加了论坛的互动性。

在这些实体之间，存在几种关键的关系：

(1) 用户和帖子之间的关系表明用户可以发布多个帖子。这是一个一对多的关系，其中单个用户可能关联多个帖子。

(2) 用户和评论之间的关系也是一对多的，表示用户可以对多个帖子进行评论。

(3) 帖子和评论之间的关系指出一个帖子可以有多个评论。这也是一个一对多的关系，表明论坛的帖子可以引发讨论，吸引多个评论。

在实际的数据库实现中，这些实体将转换为数据库表，属性将成为表中的列，实体间的关系将通过外键或关系表来实现。这样的结构设计确保了数据的组织和管理既系统又高效。

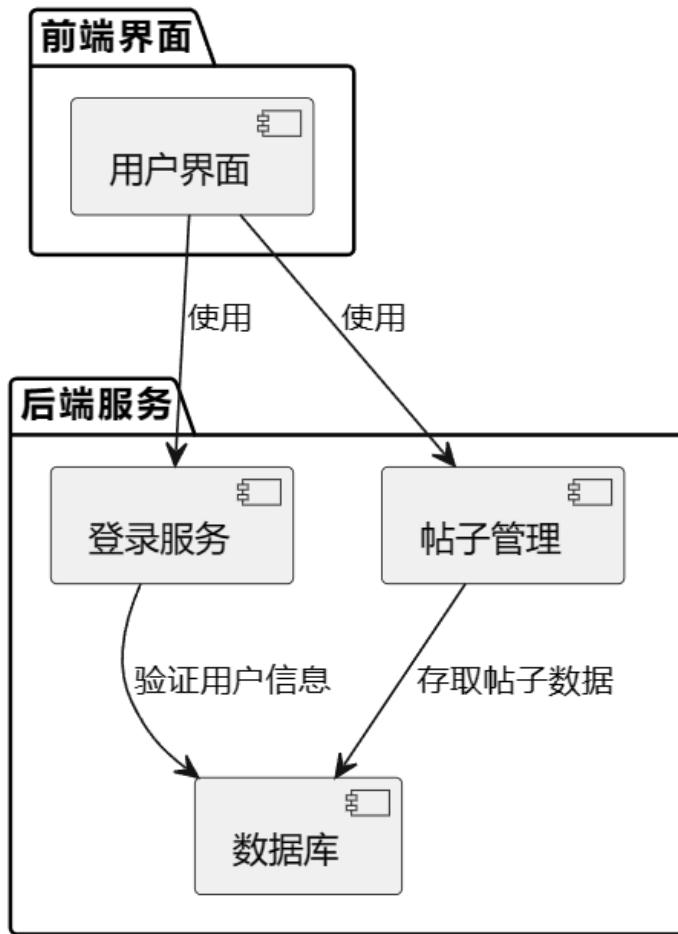


图 12: “系统架构图”

3.5 原型系统设计

对于树洞论坛系统，原型设计将包括以下关键界面和功能：

用户注册与登录界面：设计一个直观的注册页面，让新用户能够方便地创建账户，并提供一个登录页面，使得回访用户可以轻松进入论坛。

首页与帖子浏览界面：创建一个清晰的首页，展示最新或最热门的帖子列表，以及一个导航系统，允许用户根据帖子类别或关键词进行搜索和筛选。

帖子发布界面：提供一个用户友好的帖子编辑器，让用户可以编写和发布新帖子。编辑器应包括基本的文本格式工具和附件上传功能。

评论互动界面：确保帖子下方有一个区域供用户留下评论，并互动。用户应能够轻松回复他人的评论，从而促进社区内的讨论。

用户个人资料界面：允许用户查看和编辑他们的个人资料信息，如用户名、密码、联系方式等。

管理员管理界面：为管理员提供一个强大的控制面板，用于帖子审核、用户管理、统计信息查看和论坛设置等。

以下是部分原型设计界面：

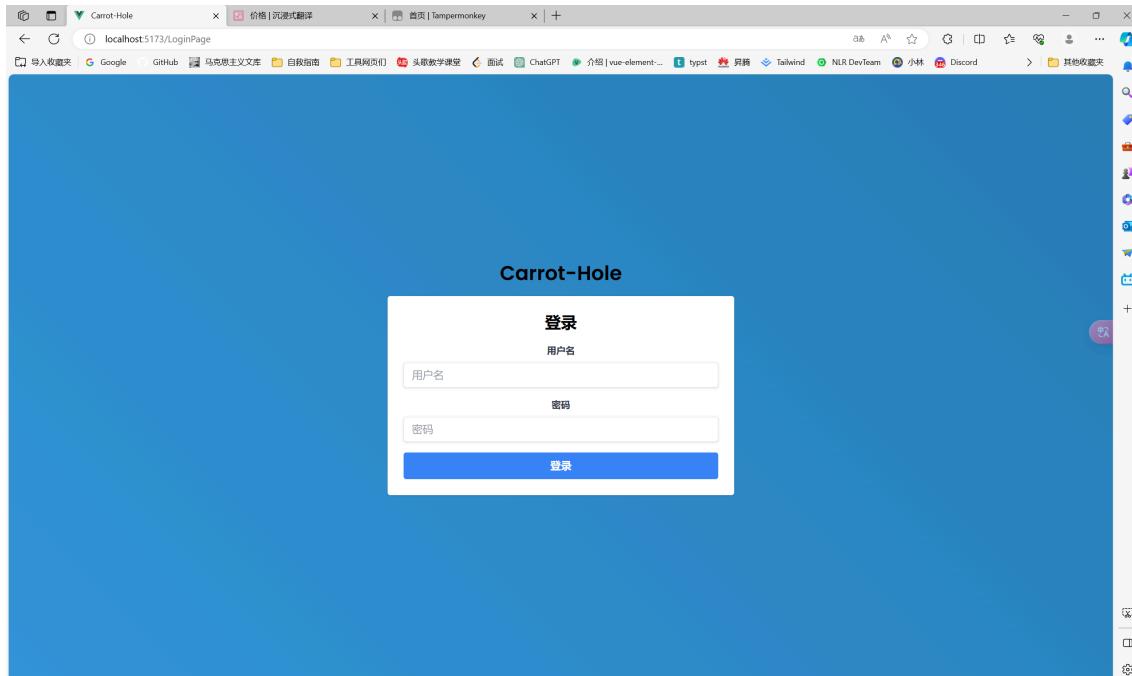


图 13: 登录界面

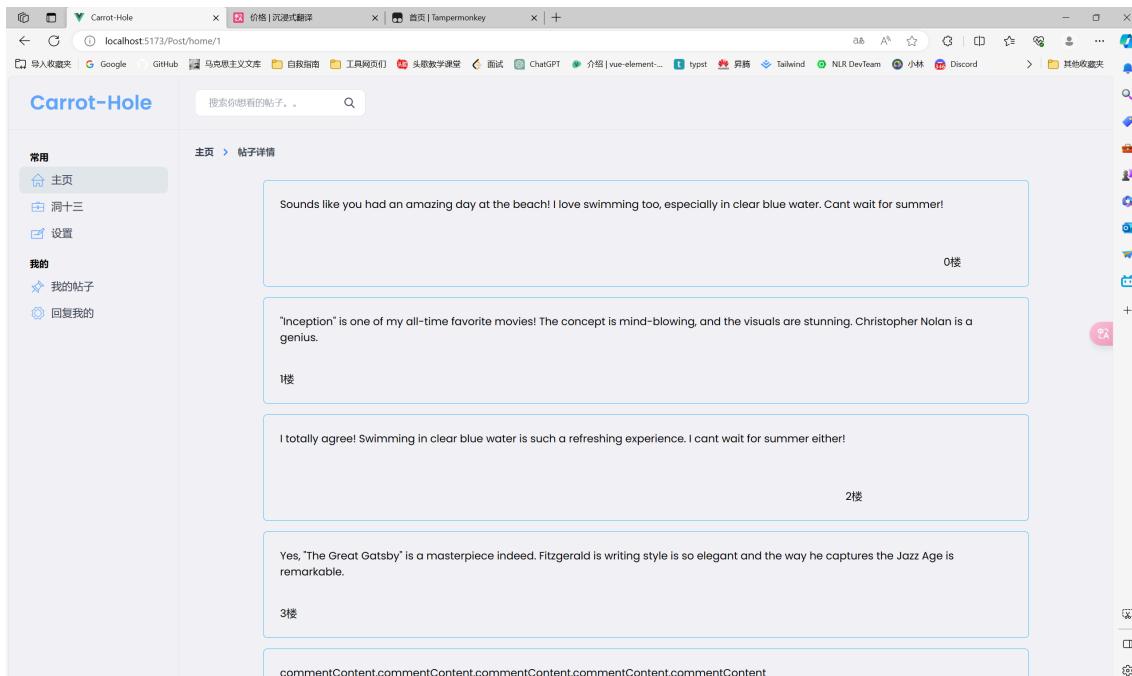


图 14: 首页

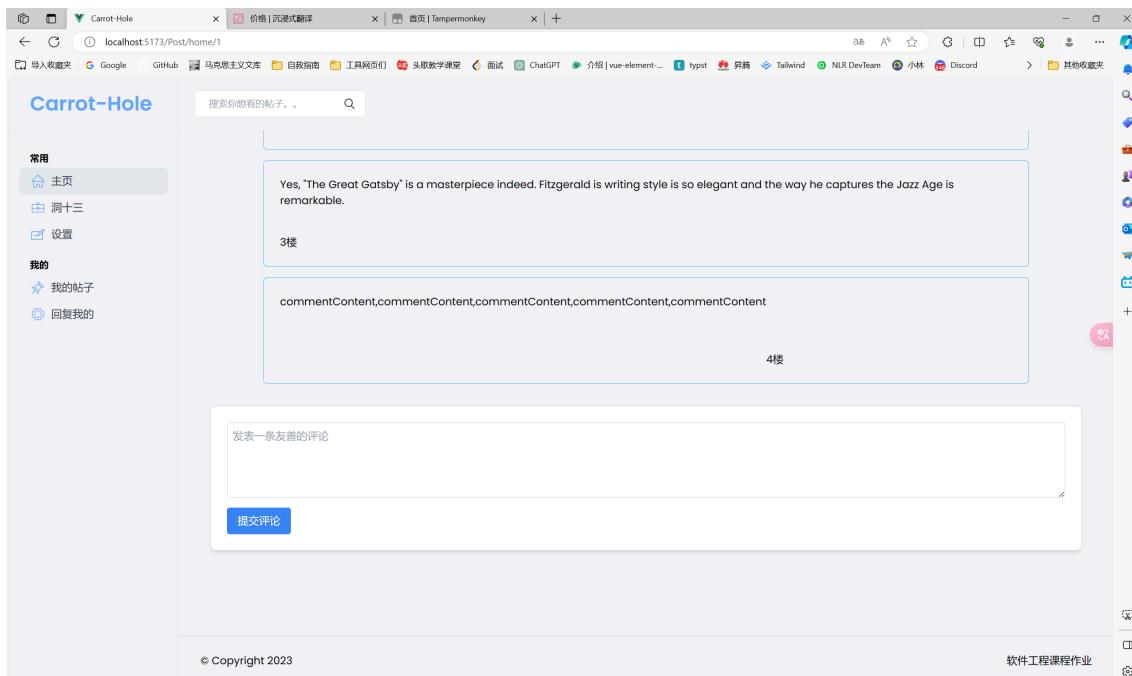


图 15: 评论页及互动页

4 概要设计和详细设计

4.1 后端

该项目采用了 Spring Boot 作为基础框架，Spring Boot 是一个用于简化 Spring 应用程序开发的框架。它通过提供默认的配置和约定大于配置的方式，使得快速构建独立、可执行的、生产级别的 Spring 应用程序变得更加容易。

在 Web 开发方面，该项目使用了 Spring MVC 框架。Spring MVC 是基于 MVC (Model-View-Controller) 设计模式的 Java Web 框架，它提供了处理请求和响应的机制，可以用于构建 RESTful 风格的 Web 应用程序。

数据库访问方面，该项目使用了 MySQL 作为数据库，并结合了 MyBatis-Plus 框架进行数据访问。MyBatis-Plus 是一个增强版的 MyBatis 框架，它提供了更多的功能和便利的开发特性，如通用 CRUD 操作、分页查询、乐观锁支持等。在该项目中，数据库托管在华为云服务器上。华为云提供了强大的云计算平台，其中包括云服务器实例。通过将数据库部署在华为云服务器上，便于统一数据，方便组内开发项目。

另外，该项目使用了 Java 17 作为项目的 Java 版本。Java 17 是 Java 编程语言的一种稳定版本，它提供了许多新的特性和改进，可以用于编写高效、现代化的 Java 应用程序。

总结起来，该项目采用了 Spring Boot 作为基础框架，使用 Spring MVC 进行 Web 开发，MySQL 作为数据库，MyBatis-Plus 作为数据访问框架，并基于 Java 17 进行开发。这些技术和工具的结合使得项目开发更加便捷和高效。

4.1.1 环境

依赖描述：

1. **spring-boot-starter-web:** 这是一个 Spring Boot 的起步依赖，用于开发 Web 应用程序。它包含了常用的 Web 开发所需的依赖和配置。
2. **spring-boot-devtools:** 这是一个开发工具，用于在开发过程中实现热部署和自动重启应用程序的功能。它提供了一些开发时的便利功能，如自动编译、重新加载等。
3. **mysql-connector-j:** 这是 MySQL 数据库的官方 Java 驱动程序，用于在 Java 应用程序中连接和操作 MySQL 数据库。

4. **spring-boot-configuration-processor:** 这个依赖用于处理 Spring Boot 应用程序中的配置文件，并生成相关的元数据，以便在 IDE 中进行代码提示和补全。
5. **lombok:** 这是一个 Java 库，用于简化 Java 代码的编写。它提供了一些注解和工具，可以自动生成一些常用的代码，如 getter 和 setter 方法、构造函数等。
6. **spring-boot-starter-test:** 这是一个用于编写和运行测试的 Spring Boot 起步依赖。它包含了常用的测试框架和工具，如 JUnit、Mockito 等。
7. **mybatis-plus-boot-starter:** 这是 MyBatis-Plus 的 Spring Boot 起步依赖，用于集成 MyBatis-Plus 框架到 Spring Boot 应用程序中。MyBatis-Plus 是一个增强版的 MyBatis 框架，提供了更多的功能和便利的开发特性。
8. **mybatis-plus-annotation:** 这是 MyBatis-Plus 的注解模块，提供了一些基于注解的开发特性，如注解查询、注解更新等。
9. **mybatis-plus-extension:** 这是 MyBatis-Plus 的扩展模块，提供了一些额外的功能和扩展点，如自定义 SQL 注入器、自定义全局配置等。
10. **mybatis-plus:** 这是 MyBatis-Plus 的核心模块，包含了基本的 MyBatis-Plus 功能和 API。

环境描述：

1. **Java 版本:** 该项目使用 Java 17 进行开发和编译。
2. **Spring Boot 版本:** 该项目使用 Spring Boot 2.7.16 作为基础框架。
3. **Maven:** 该项目使用 Maven 作为构建工具。

此外，还有一个针对 Spring Boot 的 Maven 插件配置。该插件是用于将项目打包成可执行的 JAR 文件，并提供了一些 Spring Boot 相关的功能和配置。在这个 POM 文件中，使用了 `spring-boot-maven-plugin` 插件，并配置了排除项目中的 `lombok` 依赖，以避免在打包过程中将 `lombok` 的相关代码包含进去。

4.1.2 工程配置

1. `MybatisPlusConfig.java:`

```

1 @Configuration
2 @Slf4j
3 public class MybatisPlusConfig {
4
5     /**
6      * MybatisPlus 插件配置
7      * @return {@link MybatisPlusInterceptor}

```

```

8     */
9     @Bean
10    public MybatisPlusInterceptor mybatisPlusInterceptor() {
11        MybatisPlusInterceptor interceptor = new
12        MybatisPlusInterceptor();
13        //分页插件
14        //                                         interceptor.addInnerInterceptor(new
15        PaginationInnerInterceptor(DbType.MYSQL));
16        PaginationInnerInterceptor paginationInnerInterceptor = new
17        PaginationInnerInterceptor(DbType.MYSQL);
18        //处理查询溢出，当超过最大页数时不会报错
19        paginationInnerInterceptor.setOverflow(true);
20        interceptor.addInnerInterceptor(paginationInnerInterceptor);
21        //防止全表更新与删除
22        //                                         interceptor.addInnerInterceptor(new
23        BlockAttackInnerInterceptor());
24        return interceptor;
25    }
26
27 /**
28 * 表对应的实例，自动填充创建时间和更新时间
29 */
30 @Bean
31 public MetaObjectHandler metaObjectHandler() {
32     return new MetaObjectHandler() {
33         @Override
34         public void insertFill (MetaObject metaObject){
35             log.info("auto fill-----[insert]");
36             metaObject.setValue("registerDate", LocalDateTime.now());
37             metaObject.setValue("loginDate", LocalDateTime.now());
38             metaObject.setValue("postDate", LocalDateTime.now());
39             metaObject.setValue("commentDate", LocalDateTime.now());
40         }
41         @Override
42         public void updateFill (MetaObject metaObject){
43             log.info("auto fill-----[update]");
44             metaObject.setValue("loginDate", LocalDateTime.now());
45         }
46     };
47 }
48 }
49 }
```

这是一个使用 Mybatis Plus 框架的配置类。

通过 mybatisPlusInterceptor()方法，配置了 Mybatis Plus 的插件拦截器。其中包括了分页插件（PaginationInnerInterceptor）、防止全表更新与删除的插件（BlockAttackInnerInterceptor）。

通过 metaObjectHandler()方法，配置了表对应的实例的自动填充功能。在插入数据时，自动填充创建时间（registerDate、loginDate、postDate、commentDate）；在更新数据时，自动填充登录时间（loginDate）。

2. WebConfiguration.java:

```

1  @Configuration
2  public class WebConfiguration implements WebMvcConfigurer {
3
4      @Override
5      public void addInterceptors(InterceptorRegistry registry) {
6          registry.addInterceptor(new AuthInterceptor())
7              // "/**" 表示所有请求路径都拦截，包含静态资源
8              .addPathPatterns("/**")
9              // 放行的路径
10             .excludePathPatterns("/css/**", "/img/**", "/js/**",
11             "/error/**", "/zui/**", "/login");
12     }
13 }
```

这是一个 Web 配置类，实现了 WebMvcConfigurer 接口。

通过 addInterceptors()方法，配置了拦截器（AuthInterceptor）。该拦截器用于对所有请求路径进行拦截，但排除了一些静态资源路径和登录路径。

3. application.yaml:

```

1  spring:
2      datasource:
3          driver-class-name: com.mysql.cj.jdbc.Driver
4          password: root
5          url: jdbc:mysql://1.94.32.182:3306/carrot_hole?serverTimezone=
6          UTC&useSSL=false
7          username: root
8      mybatis-plus:
9          type-aliases-package: com.hust23se.carrothole.entity # 实体类取别名
10         configuration:
11             map-underscore-to-camel-case: true
12     mybatis:
13         type-aliases-package: com.hust23se.carrothole.entity # 实体类取别名
14     # configuration:
15     #     ## 下划线转驼峰配置
16     #     map-underscore-to-camel-case: true
17     mapper-locations: classpath:mapper/*.xml
```

```

17 config-location: classpath:mybatis-config.xml
18 logging:
19   level:
20     root: info
21     com.hust23se.carrothole: debug
22   file:
23     path: logs
24 server:
25   port: 8080

```

这是一个 YAML 格式的配置文件，包含了 Spring Boot 应用的各种配置项。

在 `spring.datasource` 下配置了数据库相关信息，包括数据库驱动类、URL、用户名和密码。

在 `mybatis-plus` 和 `mybatis` 下配置了 Mybatis 和 Mybatis Plus 的相关配置项，如实体类别名、映射文件路径等。

在 `logging` 下配置了日志级别和日志文件的路径。

在 `server` 下配置了应用的端口号。

以上是对后端工程配置文件的详细描述。这些配置文件用于配置数据库、Mybatis Plus、拦截器和其他应用相关的设置，以保证后端工程的正常运行和功能实现。

4.1.3 系统结构

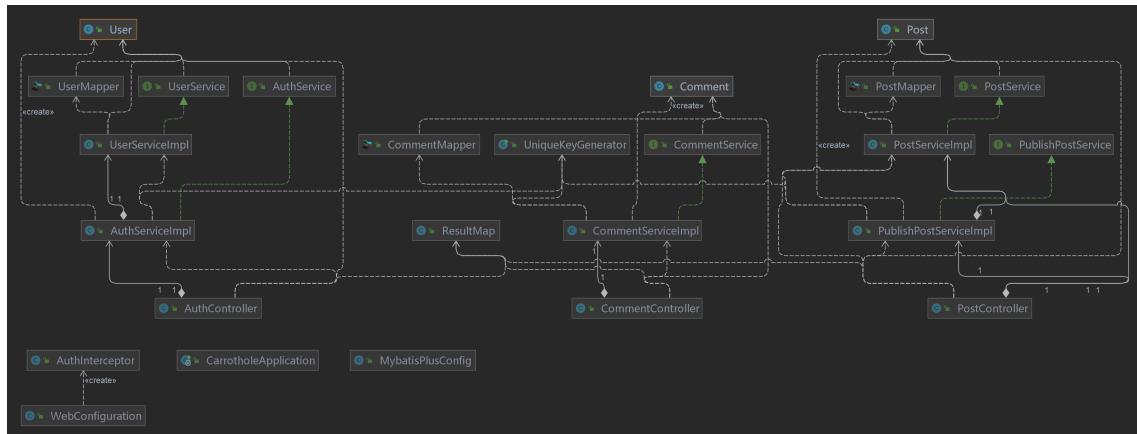


图 16: 后端类依赖图

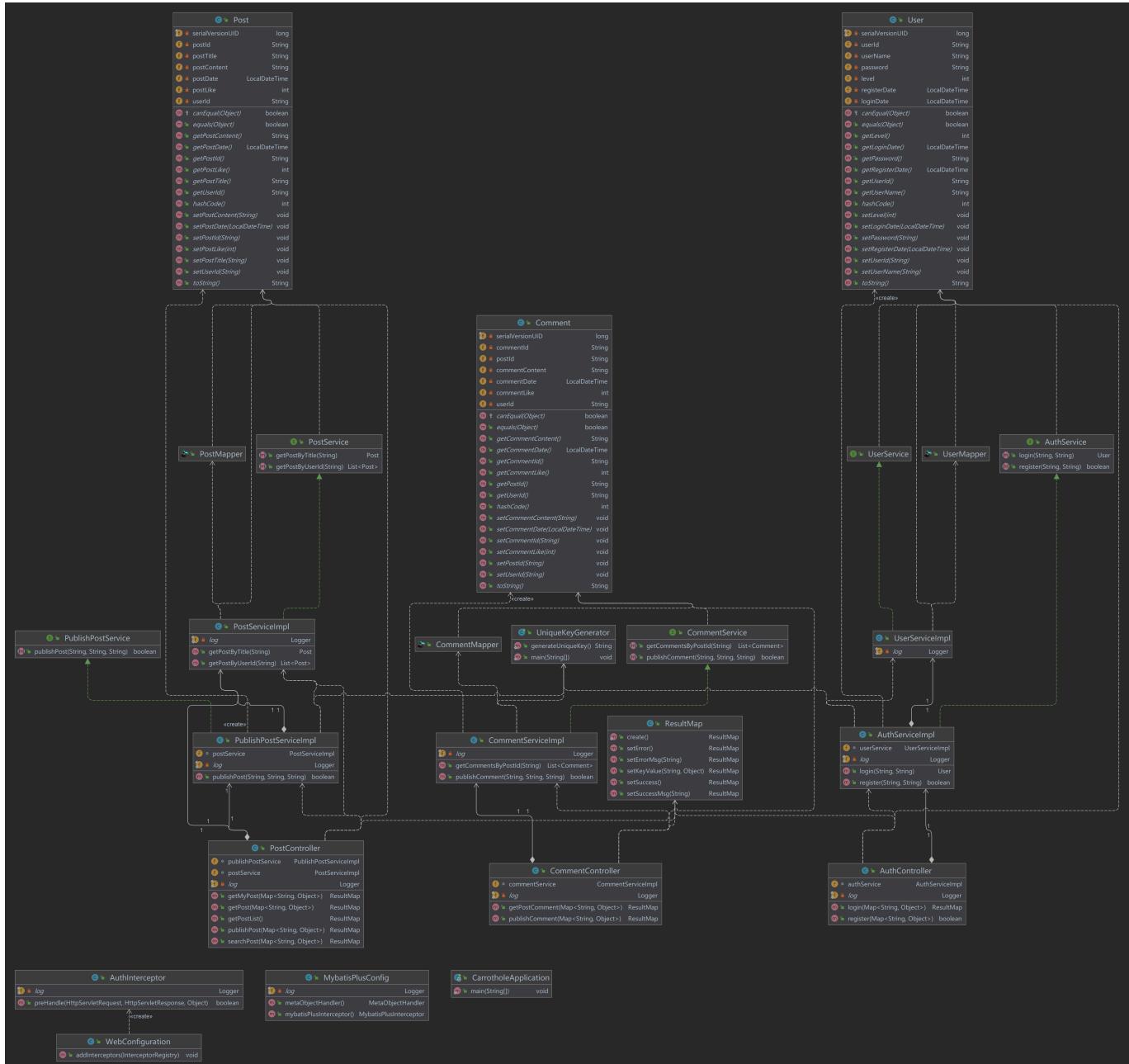


图 17: 后端系统结构图

1. Config (配置) :

- com.hust23se.carrothole.config.MybatisPlusConfig: 这个类是 Mybatis Plus 框架的配置类，用于配置 Mybatis Plus 的插件和自动填充功能。
- com.hust23se.carrothole.config.WebConfiguration: 这个类是 Web 配置类，用于配置拦截器和静态资源路径的映射。

2. Controller (控制器) :

- com.hust23se.carrothole.controller.AuthController: 这个类是处理认证相关请求的控制器。

- com.hust23se.carrothole.controller.CommentController: 这个类是处理评论相关请求的控制器。
- com.hust23se.carrothole.controller.PostController: 这个类是处理帖子相关请求的控制器。

3. Entity (实体) :

- com.hust23se.carrothole.entity.Comment: 这个类是评论实体类, 用于表示系统中的评论信息。
- com.hust23se.carrothole.entity.Post: 这个类是帖子实体类, 用于表示系统中的帖子信息。
- com.hust23se.carrothole.entity.User: 这个类是用户实体类, 用于表示系统中的用户信息。

4. Mapper (数据访问层) :

- com.hust23se.carrothole.mapper.CommentMapper: 这个类是评论的数据访问层接口, 用于定义与评论数据相关的数据库操作。
- com.hust23se.carrothole.mapper.PostMapper: 这个类是帖子的数据访问层接口, 用于定义与帖子数据相关的数据库操作。
- com.hust23se.carrothole.mapper.UserMapper: 这个类是用户的数据访问层接口, 用于定义与用户数据相关的数据库操作。

5. Service (业务逻辑层) :

- com.hust23se.carrothole.service.AuthService: 这个类是认证服务接口, 定义了与认证相关的业务逻辑。
- com.hust23se.carrothole.service.AuthServiceImpl: 这个类是认证服务的实现类, 实现了 AuthService 接口的具体业务逻辑。
- com.hust23se.carrothole.service.CommentService: 这个类是评论服务接口, 定义了与评论相关的业务逻辑。
- com.hust23se.carrothole.service.CommentServiceImpl: 这个类是评论服务的实现类, 实现了 CommentService 接口的具体业务逻辑。
- com.hust23se.carrothole.service.PostService: 这个类是帖子服务接口, 定义了与帖子相关的业务逻辑。

- com.hust23se.carrothole.service.PostServiceImpl: 这个类是帖子服务的实现类，实现了 PostService 接口的具体业务逻辑。
- com.hust23se.carrothole.service.UserService: 这个类是用户服务接口，定义了与用户相关的业务逻辑。
- com.hust23se.carrothole.service.UserServiceImpl: 这个类是用户服务的实现类，实现了 UserService 接口的具体业务逻辑。

这些类组成了系统的基本结构，用于实现用户认证、评论管理和帖子管理等功能。控制器负责接收和处理请求，服务层负责处理业务逻辑，数据访问层负责与数据库交互，实体类用于表示系统中的数据对象。通过这种结构，实现了系统的分层架构和业务逻辑的解耦，提高了系统的可维护性和可扩展性。

4.1.4 接口声明

1. AuthController

```

1  @RestController
2  @RequestMapping("/auth")
3  public class AuthController {
4
5      @Autowired
6      AuthServiceImpl authService;
7
8      @PostMapping("/register")
9      public boolean register(@RequestBody Map<String, Object> idMap)
10     throws Exception {
11         try{
12             return authService.register(String.valueOf(idMap.get("userName")), String.valueOf(idMap.get("password")));
13         }catch (Exception e){
14             throw new Exception("register error");
15         }
16     }
17
18     @PostMapping("/login")
19     public ResultMap login(@RequestBody Map<String, Object> idMap)
20     throws Exception{
21         try{
22             User user = authService.login(String.valueOf(idMap.get("userName")), String.valueOf(idMap.get("password")));
23             ResultMap resultMap = ResultMap.create();
24             if(user != null){
25                 resultMap.setSuccess();
26                 resultMap.setSuccessMsg("Login success");
27                 resultMap.setKeyValue("user", user);
28             }else{
29                 resultMap.setError();
30                 resultMap.setErrorMsg("Login failed");
31             }
32         }catch (Exception e){
33             throw new Exception("login error");
34         }
35     }
36
37 }
```

```

29
30
31     return resultMap;
32 }catch (Exception e){
33     throw new Exception("login error");
34 }
35 }
36 }
```

该控制器类使用@RequestMapping(“/auth”)注解来映射根 URL 路径，即所有该控制器处理的请求都应该以/auth 开头。

下面是该控制器类中的两个请求处理方法：

- register 方法：

使用 @PostMapping(“/register”)注解将该方法映射到/auth/register 路径的 POST 请求。

该方法接受一个 Map<String, Object>类型的请求体（通过@RequestBody 注解），其中包含了用户名和密码信息。

方法内部调用了一个名为 authService 的 AuthServiceImpl 对象的 register 方法，将用户名和密码作为参数传递给它。

如果 register 方法返回 true，则表示注册成功，该方法将返回 true 作为响应。

如果 register 方法抛出异常，则将异常信息封装为 Exception 对象并抛出。

- login 方法：

使用@PostMapping(“/login”)注解将该方法映射到/auth/login 路径的 POST 请求。

该方法接受一个 Map<String, Object>类型的请求体（通过@RequestBody 注解），其中包含了用户名和密码信息。

方法内部调用了一个名为 authService 的 AuthServiceImpl 对象的 login 方法，将用户名和密码作为参数传递给它。

如果 login 方法返回一个非空的 User 对象，表示登录成功。

在登录成功的情况下，该方法创建了一个 ResultMap 对象，并设置了成功的状态码和消息，并将 User 对象作为值存储在 ResultMap 中的 user 键下。

在登录失败的情况下，该方法创建了一个 ResultMap 对象，并设置了错误的状态码和消息。

最后，该方法将创建的 ResultMap 对象作为响应返回。

如果 login 方法抛出异常，则将异常信息封装为 Exception 对象并抛出。

2. PostController

```

1  @RestController
2  @RequestMapping("/post")
3  public class PostController {
4
5      @Autowired
6      PublishPostServiceImpl publishPostService;
7
8      @Autowired
9      PostServiceImpl postService;
10
11     @PostMapping("/publish")
12     public ResultMap publishPost(@RequestBody Map<String, Object>
13         idMap) throws Exception {
14         try{
15             if(publishPostService.publishPost(String.valueOf(idMap.get("postTitle"))),Str
16
17             return
18             ResultMap.create().setSuccess().setSuccessMsg("publish success");
19         }
20         return ResultMap.create().setError().setErrorMsg("publish
21 failed");
22     }catch (Exception e){
23         throw new Exception("publish post error");
24     }
25
26     @PostMapping("/get")
27     public ResultMap getPost(@RequestBody Map<String, Object> idMap)
28     throws Exception{
29         try{
30             Post      post      =
31             postService.getById(String.valueOf(idMap.get("postId")));
32             if(post == null){
33                 return ResultMap.create().setError().setErrorMsg("Not
34 found");
35             }else{
36                 return ResultMap.create().setSuccess().setSuccessMsg("Success").setKeyValue
37             }
38         }catch (Exception e){
39             throw new Exception("get post error");
40         }
41
42     @PostMapping("/search")
43     public ResultMap searchPost(@RequestBody Map<String, Object> idMap)
44     throws Exception {

```

```

39     try{
40         Post      post      =
41         postService.getPostByTitle(String.valueOf(idMap.get("postTitle")));
42         if(post == null){
43             return ResultMap.create().setError().setErrorMsg("Not
44             found");
45         }else{
46             return ResultMap.create().setSuccess().setSuccessMsg("Success").setKeyValue(
47                 );
48         }
49     }
50
51     @GetMapping("/getPostList")
52     public ResultMap getPostList() throws Exception{
53         try{
54             List<Post> postList= postService.list();
55             return ResultMap.create().setSuccess().setSuccessMsg("Success").setKeyValue(
56                 );
57         }catch (Exception e){
58             throw new Exception("search post list error");
59         }
60
61     @PostMapping("/getMyPost")
62     public ResultMap getMyPost(@RequestBody Map<String, Object> idMap)
63     throws Exception{
64         try{
65             List<Post>      postList=
66             postService.getPostByUserId(String.valueOf(idMap.get("userId")));
67             return ResultMap.create().setSuccess().setSuccessMsg("Success").setKeyValue(
68                 );
69         }catch (Exception e){
70             throw new Exception("search post list error");
71         }
72     }
73 }
```

该控制器类使用@RequestMapping(“/post”)注解来映射根 URL 路径，即所有该控制器处理的请求都应该以/post 开头。

下面是该控制器类中的几个请求处理方法：

- publishPost 方法：

使用 @PostMapping(“/publish”)注解将该方法映射到/post/publish 路径的 POST 请求。

该方法接受一个 Map<String, Object>类型的请求体（通过@RequestBody 注解），其中包含了发布帖子所需的标题、内容和用户 ID 信息。

方法内部调用了一个名为 publishPostService 的 PublishPostServiceImpl 对象的 publishPost 方法，将标题、内容和用户 ID 作为参数传递给它。

如果 publishPost 方法返回 true，表示发布成功，该方法将返回一个包含成功状态和消息的 ResultMap 对象。

如果 publishPost 方法返回 false，表示发布失败，该方法将返回一个包含错误状态和消息的 ResultMap 对象。

如果 publishPost 方法抛出异常，则将异常信息封装为 Exception 对象并抛出。

- **getPost 方法：**

使用 @PostMapping(“/get”)注解将该方法映射到/post/get 路径的 POST 请求。

该方法接受一个 Map<String, Object>类型的请求体（通过@RequestBody 注解），其中包含了需要获取的帖子的 ID 信息。

方法内部调用了一个名为 postService 的 PostServiceImpl 对象的 getById 方法，将帖子 ID 作为参数传递给它。

如果 getById 方法返回一个非空的 Post 对象，表示获取成功。

在获取成功的情况下，该方法创建了一个 ResultMap 对象，并设置了成功的状态码和消息，并将 Post 对象作为值存储在 ResultMap 中的 post 键下。

在获取失败的情况下，该方法创建了一个 ResultMap 对象，并设置了错误的状态码和消息。

最后，该方法将创建的 ResultMap 对象作为响应返回。

如果 getById 方法抛出异常，则将异常信息封装为 Exception 对象并抛出。

- **searchPost 方法：**

使用 @PostMapping(“/search”)注解将该方法映射到/post/search 路径的 POST 请求。

该方法接受一个 Map<String, Object>类型的请求体（通过@RequestBody 注解），其中包含了需要搜索的帖子的标题信息。

方法内部调用了一个名为 postService 的 PostServiceImpl 对象的 getPostByTitle 方法，将帖子标题作为参数传递给它。

如果 getPostByTitle 方法返回一个非空的 Post 对象，表示搜索成功。

在搜索成功的情况下，该方法创建了一个 ResultMap 对象，并设置了成功的状态码和消息，并将 Post 对象作为值存储在 ResultMap 中的 post 键下。

在搜索失败的情况下，该方法创建了一个 ResultMap 对象，并设置了错误的状态码和消息。

最后，该方法将创建的 ResultMap 对象作为响应返回。

如果 `getPostByTitle` 方法抛出异常，则将异常信息封装为 `Exception` 对象并抛出。

- `getPostList` 方法：

使用`@GetMapping("/getPostList")`注解将该方法映射到`/post/getPostList`路径的 GET 请求。

该方法不接受请求体，直接返回帖子列表。

方法内部调用了一个名为 `postService` 的 `PostServiceImpl` 对象的 `list` 方法，获取所有帖子的列表。

如果 `list` 方法返回一个非空的帖子列表，表示获取成功。

在获取成功的情况下，该方法创建了一个 ResultMap 对象，并设置了成功的状态码和消息，并将帖子列表作为值存储在 ResultMap 中的 `postList` 键下。

在获取失败的情况下，该方法创建了一个 ResultMap 对象，并设置了错误的状态码和消息。

最后，该方法将创建的 ResultMap 对象作为响应返回。

如果 `list` 方法抛出异常，则将异常信息封装为 `Exception` 对象并抛出。

- `getMyPost` 方法：

使用`@PostMapping("/getMyPost")`注解将该方法映射到`/post/getMyPost`路径的 POST 请求。

该方法接受一个 `Map<String, Object>`类型的请求体（通过`@RequestBody`注解），其中包含了需要获取的用户的 ID 信息。

方法内部调用了一个名为 `postService` 的 `PostServiceImpl` 对象的 `getPostByUserId` 方法，将用户 ID 作为参数传递给它。

该方法返回该用户发布的帖子列表。

如果 `getPostByUserId` 方法返回一个非空的帖子列表，表示获取成功。

在获取成功的情况下，该方法创建了一个 ResultMap 对象，并设置了成功的状态码和消息，并将帖子列表作为值存储在 ResultMap 中的 `postList` 键下。

在获取失败的情况下，该方法创建了一个 ResultMap 对象，并设置了错误的状态码和消息。

最后，该方法将创建的 ResultMap 对象作为响应返回。

如果 getPostByUserId 方法抛出异常，则将异常信息封装为 Exception 对象并抛出。

3. CommentController

```

1  @RestController
2  @RequestMapping("/comment")
3  public class CommentController {
4
5      @Autowired
6      CommentServiceImpl commentService;
7
8      @PostMapping("/publish")
9      public ResultMap publishComment(@RequestBody Map<String, Object>
10         idMap) throws Exception {
11          try{
12              if(commentService.publishComment(String.valueOf(idMap.get("postId")), String.
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29 } }
```

该控制器类使用@RequestMapping(“/comment”)注解将根 URL 路径映射为/comment，即所有该控制器处理的请求都应该以/comment 开头。

以下是该控制器类中的几个请求处理方法：

- publishComment 方法：

使用@PostMapping(“/publish”)注解将该方法映射到/comment/publish 路径的 POST 请求。

该方法接受一个 Map<String, Object>类型的请求体（通过@RequestBody 注解），其中包含了发布评论所需的帖子 ID、用户 ID 和评论内容信息。

方法内部调用了一个名为 commentService 的 CommentServiceImpl 对象的 publishComment 方法，将帖子 ID、用户 ID 和评论内容作为参数传递给它。

如果 publishComment 方法返回 true，表示评论发布成功，该方法将返回一个包含成功状态和消息的 ResultMap 对象。

如果 publishComment 方法返回 false，表示评论发布失败，该方法将返回一个包含错误状态和消息的 ResultMap 对象。

如果 publishComment 方法抛出异常，则将异常信息封装为 Exception 对象并抛出。

- getPostComment 方法：

使用@PostMapping(“/getPostComment”)注解将该方法映射到/comment/getPostComment 路径的 POST 请求。

该方法接受一个 Map<String, Object>类型的请求体（通过@RequestBody 注解），其中包含了需要获取评论的帖子 ID 信息。

方法内部调用了一个名为 commentService 的 CommentServiceImpl 对象的 getCommentsByPostId 方法，将帖子 ID 作为参数传递给它。

getCommentsByPostId 方法返回一个包含了该帖子下所有评论的 Comment 对象的列表。

该方法创建了一个 ResultMap 对象，并设置了成功的状态码和消息，同时将帖子 ID 和评论列表存储在 ResultMap 中的相应键下。

如果 getCommentsByPostId 方法抛出异常，则将异常信息封装为 Exception 对象并抛出。

4.1.5 服务实现

1. AuthService

```

1 public interface AuthService {
2
3     /**
4      * register new user

```

```

5   * @param userName
6   * @param password
7   * @return
8   */
9   public boolean register(String userName, String password);
10
11 /**
12  * check user exist
13  * @param userName
14  * @param password
15  * @return
16  */
17 public User login(String userName, String password);
18 }

```

```

1 @Service
2 @Slf4j
3 public class AuthServiceImpl implements AuthService{
4
5     @Autowired
6     UserServiceImpl userService;
7
8 /**
9  * register new user
10 *
11 * @param userName
12 * @param password
13 * @return
14 */
15 @Override
16 public boolean register(String userName, String password) {
17     QueryWrapper<User> userQueryWrapper = new QueryWrapper<>();
18     userQueryWrapper.eq("user_name",userName);
19     // repeated userName
20     if(userService.getOne(userQueryWrapper) != null){
21         return false;
22     }
23     User user = new User();
24     user.setUserId(UniqueKeyGenerator.generateUniqueKey());
25     user.setUserName(userName);
26     user.setPassword(password);
27     user.setRegisterDate(LocalDateTime.now());
28     user.setLoginDate(LocalDateTime.now());
29     user.setLevel(1);
30     return userService.save(user);
31 }
32
33 /**
34  * check user exist
35  *
36  * @param userName
37  * @param password
38  * @return

```

```

39     */
40     @Override
41     public User login(String userName, String password) {
42         QueryWrapper<User> userQueryWrapper = new QueryWrapper<>();
43
44         userQueryWrapper.eq("user_name",userName).eq("password",password);
45         User user = userService.getOne(userQueryWrapper);
46         if(user !=null){
47             user.setPassword(null);
48             user.setLoginDate(LocalDateTime.now());
49         }
50     }
51 }

```

- 接口定义:

AuthService 接口定义了两个方法: register 和 login。register 用于注册新用户, login 用于验证用户登录。

- AuthServiceImpl 类:

AuthServiceImpl 类是 AuthService 接口的实现类。该类使用了@Service 注解, 表示它是一个服务组件, 可以被其他组件注入和使用。使用@Slf4j 注解, 用于生成日志记录。

- register 方法:

register 方法用于注册新用户。

首先, 使用 QueryWrapper 对象构建一个查询条件, 根据用户名查询用户是否已存在。如果存在重复的用户名, 返回 false, 表示注册失败。如果用户名不存在重复, 创建一个新的 User 对象, 并设置用户的各个属性, 如用户 ID、用户名、密码、注册日期和登录日期等。调用 userService 的 save 方法将用户信息保存到数据库中, 并返回保存操作的结果。

- login 方法:

login 方法用于验证用户登录。

同样使用 QueryWrapper 对象构建查询条件, 根据用户名和密码查询用户。如果查询到了用户, 则将用户的密码字段设为 null, 以避免将密码返回给调用方。更新用户的登录日期为当前日期时间。返回查询到的用户对象, 如果用户不存在, 则返回 null。

这是一个简单的身份验证服务实现，通过调用 register 方法进行用户注册，调用 login 方法进行用户登录验证。实现类使用了 MyBatis-Plus 框架提供的 QueryWrapper 对象进行数据库查询操作。

2. CommentService

```

1 public interface CommentService extends IService<Comment> {
2
3     public boolean publishComment(String postId, String userId, String
4         commentContent);
5
6     public List<Comment> getCommentsByPostId(String postId);
7 }
```

```

1 @Service("CommentServiceImpl")
2 @Slf4j
3 public class CommentServiceImpl extends ServiceImpl<CommentMapper,
4     Comment> implements CommentService{
5
6     @Override
7     public boolean publishComment(String postId, String userId, String
8         commentContent) {
9         Comment comment = new Comment();
10        comment.setCommentId(UniqueKeyGenerator.generateUniqueKey());
11        comment.setCommentContent(commentContent);
12        comment.setCommentLike(0);
13        comment.setCommentDate(LocalDateTime.now());
14        comment.setUserId(userId);
15        comment.setPostId(postId);
16        return this.save(comment);
17    }
18
19    @Override
20    public List<Comment> getCommentsByPostId(String postId) {
21        QueryWrapper<Comment> commentQueryWrapper = new
22        QueryWrapper<>();
23        commentQueryWrapper.eq("post_id", postId);
24        return this.list(commentQueryWrapper);
25    }
26 }
```

- 接口定义：

CommentService 接口定义了两个方法：publishComment 和 getCommentsByPostId。publishComment 用于发布评论，getCommentsByPostId 用于根据帖子 ID 获取评论列表。

- CommentServiceImpl 类：

CommentServiceImpl 类是 CommentService 接口的实现类。使用@Service 注解，表示它是一个服务组件，可以被其他组件注入和使用。类继承了 ServiceImpl<CommentMapper, Comment>，这个类是 MyBatis-Plus 提供的通用 Service 实现类，提供了基本的增删改查操作的默认实现。

- publishComment 方法：

publishComment 方法用于发布评论。

首先，创建一个 Comment 对象，并设置评论的各个属性，如评论 ID、评论内容、评论点赞数、评论日期、用户 ID 和帖子 ID 等。调用 this.save 方法将评论信息保存到数据库中，并返回保存操作的结果。

- getCommentsByPostId 方法：

getCommentsByPostId 方法用于根据帖子 ID 获取评论列表。

创建一个 QueryWrapper 对象，根据帖子 ID 查询对应的评论。调用 this.list 方法执行查询操作，并返回查询到的评论列表。

这是一个简单的评论服务实现，通过调用 publishComment 方法发布评论，调用 getCommentsByPostId 方法获取帖子的评论列表。实现类继承了 MyBatis-Plus 提供的 ServiceImpl 类，利用其提供的默认实现来实现基本的增删改查操作。

3. PostService

```

1 public interface PostService extends IService<Post> {
2
3     public Post getPostByTitle(String title);
4
5     public List<Post> getPostByUserId(String userId);
6 }
```

```

1 @Service("PostServiceImpl")
2 @Slf4j
3 public class PostServiceImpl extends ServiceImpl<PostMapper, Post>
4     implements PostService{
5
6     @Override
7     public Post getPostByTitle(String title) {
8         QueryWrapper<Post> postQueryWrapper = new QueryWrapper<>();
9         postQueryWrapper.eq("post_title", title);
10        return this.getOne(postQueryWrapper);
11    }
12
13    @Override
14    public List<Post> getPostByUserId(String userId) {
```

```

14     QueryWrapper<Post> postQueryWrapper = new QueryWrapper<>();
15     postQueryWrapper.eq("user_id",userId);
16     return this.list(postQueryWrapper);
17 }
18 }
```

- 接口定义：

PostService 接口定义了两个方法：getPostByTitle 和 getPostByUserId。getPostByTitle 用于根据标题获取帖子，getPostByUserId 用于根据用户 ID 获取帖子列表。

- PostServiceImpl 类：

PostServiceImpl 类是 PostService 接口的实现类。使用@Service 注解，表示它是一个服务组件，可以被其他组件注入和使用。使用@Slf4j 注解，用于生成日志记录。类继承了 ServiceImpl<PostMapper, Post>，这个类是 MyBatis-Plus 提供的通用 Service 实现类，提供了基本的增删改查操作的默认实现。

- getPostByTitle 方法：

getPostByTitle 方法用于根据标题获取帖子。

创建一个 QueryWrapper 对象，根据标题查询对应的帖子。调用 this.getOne 方法执行查询操作，并返回查询到的帖子。

- getPostByUserId 方法：

getPostByUserId 方法用于根据用户 ID 获取帖子列表。

创建一个 QueryWrapper 对象，根据用户 ID 查询对应的帖子。调用 this.list 方法执行查询操作，并返回查询到的帖子列表。

这是一个简单的帖子服务实现，通过调用 getPostByTitle 方法根据标题获取帖子，调用 getPostByUserId 方法根据用户 ID 获取帖子列表。实现类继承了 MyBatis-Plus 提供的 ServiceImpl 类，利用其提供的默认实现来实现基本的增删改查操作。

4. PublishPostService

```

1 public interface PublishPostService {
2     public boolean publishPost(String postTitle, String
3     PostContent, String userId);
4 }
```

```

1 @Service
2 @Slf4j
3 public class PublishPostServiceImpl implements PublishPostService{
4 }
```

```

5     @Autowired
6     PostServiceImpl postService;
7
8     @Override
9         public boolean publishPost(String postTitle, String
10    postContent, String userId) {
11        Post post = new Post();
12        post.setPostId(UniqueKeyGenerator.generateUniqueKey());
13        post.setPostDate(LocalDateTime.now());
14        post.setPostLike(0);
15        post.setPostTitle(postTitle);
16        post.setPostContent(postContent);
17        post.setUserId(userId);
18        return postService.save(post);
19    }

```

- 接口定义：

PublishPostService 接口定义了一个方法：publishPost，用于发布帖子。

- PublishPostServiceImpl 类：

PublishPostServiceImpl 类是 PublishPostService 接口的实现类。使用@Service 注解，表示它是一个服务组件，可以被其他组件注入和使用。使用@Autowired 注解将 PostServiceImpl 注入到 PublishPostServiceImpl 中，以便调用 PostService 的方法。

- publishPost 方法：

publishPost 方法用于发布帖子。

首先，创建一个 Post 对象，并设置帖子的各个属性，如帖子 ID、帖子日期、帖子点赞数、帖子标题、帖子内容和用户 ID 等。调用 postService 的 save 方法将帖子信息保存到数据库中，并返回保存操作的结果。

这是一个简单的发布帖子服务实现，通过调用 publishPost 方法将帖子信息保存到数据库中。实现类使用了@Autowired 注解将 PostServiceImpl 注入，以便调用 PostService 的方法。

4.1.6 数据结构

1. User

```

1 @Data
2 @NoArgsConstructor
3 @TableName("user")

```

```

4  public class User implements Serializable {
5
6      @Serial
7      private static final long serialVersionUID = 1L;
8
9      @TableId(value = "user_id")
10     private String userId;
11
12     @TableField("user_name")
13     private String userName;
14
15     @TableField("password")
16     private String password;
17
18     /**
19      * user level
20      * 0 represents administrator level
21      * 1 represents common user level
22      */
23     @TableField("level")
24     private int level;
25
26     @TableField("register_date")
27     private LocalDateTime registerDate;
28
29     @TableField("login_date")
30     private LocalDateTime loginDate;
31
32 }

```

```

1  create table user
2  (
3      user_id      varchar(50) not null
4          primary key,
5      user_name    varchar(50) null,
6      password     varchar(50) null,
7      level        int         null,
8      register_date date       null,
9      login_date   date       null
10 );

```

- **user_id:** 这是一个字符串类型的字段，最大长度为 50。它是用户表中的主键，用于唯一标识每个用户。主键是一个唯一的标识符，确保每个用户在表中都有唯一的标识。
- **user_name:** 这是一个字符串类型的字段，最大长度为 50。它表示用户的名称，可以用来存储用户的真实姓名或用户名。
- **password:** 这是一个字符串类型的字段，最大长度为 50。它用于存储用户的密码，以确保用户的数据安全。

- **level:** 这是一个整数类型的字段，用于表示用户的级别。级别字段可以用来区分不同类型的用户，例如管理员和普通用户。在这个数据结构中，级别为 0 表示管理员级别，级别为 1 表示普通用户级别。
- **register_date:** 这是一个日期类型的字段，用于存储用户的注册日期。它记录了用户注册到系统的日期和时间。
- **login_date:** 这是一个日期类型的字段，用于存储用户的登录日期。每当用户登录到系统时，该字段将记录登录的日期和时间。

通过这些字段，“User”数据结构可以存储用户的基本信息，如用户 ID、用户名、密码、用户级别以及注册和登录日期。这个数据结构可以用于在数据库中创建一个名为“user”的表格，并提供方便的操作方法来插入、更新、删除和查询用户数据。

2. Post

```

1  @Data
2  @NoArgsConstructor
3  @TableName("post")
4  public class Post implements Serializable {
5
6      @Serial
7      private static final long serialVersionUID = 1L;
8
9      @TableId("post_id")
10     private String postId;
11
12     @TableField("post_title")
13     private String postTitle;
14
15     @TableField("post_content")
16     private String postContent;
17
18     @TableField("post_date")
19     private LocalDateTime postDate;
20
21     @TableField("post_like")
22     private int postLike;
23
24     @TableField("user_id")
25     private String userId;
26 }
```

```

1  create table post
2  (
3      post_id      varchar(50)  not null
4          primary key,
5      post_title   varchar(255) null,
6      post_content text        null,
```

```

7     post_date    date      null,
8     post_like    int       null,
9     user_id     varchar(50) not null
10 );

```

- **post_id:** 这是一个字符串类型的字段，最大长度为 50。它是帖子表中的主键，用于唯一标识每篇帖子。主键是一个唯一的标识符，确保每篇帖子在表中都有唯一的标识。
- **post_title:** 这是一个字符串类型的字段，最大长度为 255。它表示帖子的标题，用于描述帖子的主题或内容概要。
- **post_content:** 这是一个文本类型的字段，用于存储帖子的详细内容。文本类型字段可以容纳较长的文本数据，适合存储帖子的正文、评论等信息。
- **post_date:** 这是一个日期时间类型的字段，用于记录帖子的发布日期和时间。它指示了帖子何时被创建或发布。
- **post_like:** 这是一个整数类型的字段，用于表示帖子的点赞数。它记录了帖子被点赞的次数，可以用于衡量帖子的受欢迎程度或用户对帖子的喜爱程度。
- **user_id:** 这是一个字符串类型的字段，最大长度为 50。它表示发帖用户的 ID，用于关联帖子与其对应的用户。通过该字段，可以知道哪个用户发布了该篇帖子。通过这些字段，“Post”数据结构可以存储帖子的相关信息，如帖子 ID、标题、内容、发布日期、点赞数和发帖用户 ID。这个数据结构可以用于在数据库中创建一个名为“-post”的表格，并提供方便的操作方法来插入、更新、删除和查询帖子数据。

3. Comment

```

1  @Data
2  @NoArgsConstructor
3  @TableName("comment")
4  public class Comment implements Serializable {
5
6      @Serial
7      private static final long serialVersionUID = 1L;
8
9      @TableId("comment_id")
10     private String commentId;
11
12     @TableField("post_id")
13     private String postId;
14
15     @TableField("comment_content")
16     private String commentContent;
17

```

```

18     @TableField("comment_date")
19     private LocalDateTime commentDate;
20
21     @TableField("comment_like")
22     private int commentLike;
23
24     @TableField("user_id")
25     private String userId;
26 }
```

```

1 create table comment
2 (
3     comment_id      varchar(50) not null
4         primary key,
5     post_id        varchar(50) null,
6     comment_content text        null,
7     comment_date    date        null,
8     comment_like    int         null,
9     user_id        varchar(50) not null
10 );
```

- **comment_id:** 这是一个字符串类型的字段，最大长度为 50。它是评论表中的主键，用于唯一标识每条评论。主键是一个唯一的标识符，确保每条评论在表中都有唯一的标识。
- **post_id:** 这是一个字符串类型的字段，最大长度为 50。它表示被评论的帖子的 ID。通过这个字段，可以将评论与对应的帖子关联起来，知道哪个帖子收到了哪些评论。
- **comment_content:** 这是一个文本类型的字段，用于存储评论的内容。文本类型字段可以容纳较长的文本数据，适合存储评论的正文信息。
- **comment_date:** 这是一个日期类型的字段，用于记录评论的发布日期。它指示了评论何时被创建或发布。
- **comment_like:** 这是一个整数类型的字段，用于表示评论的点赞数。它记录了评论被点赞的次数，可以用于衡量评论的受欢迎程度或用户对评论的喜爱程度。
- **user_id:** 这是一个字符串类型的字段，最大长度为 50。它表示发表评论的用户的 ID，用于关联评论与其对应的用户。通过该字段，可以知道哪个用户发布了该条评论。

通过这些字段，“Comment”数据结构可以存储评论的相关信息，如评论 ID、所评论的帖子 ID、内容、发布日期、点赞数和发表评论的用户 ID。这个数据结构可以用

于在数据库中创建一个名为"comment"的表格，并提供方便的操作方法来插入、更新、删除和查询评论数据。

4.2 前端

项目使用 vue 作为项目的前端框架，nginx 作为服务转发服务器。

4.2.1 系统结构

首先给出项目前端的项目结构：

```
1 卷 software 的文件夹 PATH 列表
2 卷序列号为 92E1-68FE
3 D:.
4   .eslintrc.json
5   index.html
6   package-lock.json
7   package.json
8   postcss.config.js
9   README.md
10  tailwind.config.js
11  vite.config.js
12
13  .vite
14
15
16  public
17    favicon.ico
18    index.html
19
20    resources
21      logo.webp
22
23  src
24    App.vue
25    main.js
26
27    assets
28      homepage.svg
29      logo.svg
30
31    css
32      Background.module.css
33      tailwind.css
34
35    components
36      Footer.vue
37      FullPoster.vue
38      HomePage.vue
39      LoginPage.vue
40      Navbar.vue
41      PreviewPoster.vue
```

```

42      Sidebar.vue
43
44  pages
45      Home.vue
46      LoginPage.vue
47      Post.vue
48      PostPage.vue
49      ReleasePost.vue
50      test.vue
51
52  router
53      index.js
54
55  store
56      index.js
57
58  utils
59      axios.js
60      event.js
61      post.js
62      request.js
63      接口.md

```

项目结构

src 中 pages 下是页面模块，utils 是 http 请求和其他处理函数，router 下是页面的逻辑和跳转，store 下是状态逻辑，components 下是侧边栏和底部栏之类不需要跟着主页面切换切换的 bar，assets 里存储 logo，css 之类静态文件。

前端的依赖使用 npm 包管理器进行管理，在根目录下使用 npm install 来读取 package.json 并下载依赖 以下是项目使用的主要依赖和对应功能：

运行时依赖项

1. v-md-editor (^2.3.17)

一个支持 Vue 3 的 Markdown 编辑器

2. axios (^1.5.1)

用于进行 HTTP 请求的 Promise based 的库

3. chart.js (^4.4.0)

一个强大且灵活的图表库

4. element-plus (^2.4.1)

基于 Element UI 的 Vue 3 组件库

5. vue (^3.3.4)

Vue.js 框架的核心库

6. vue-router (^4.2.5)

Vue.js 应用的官方路由管理器

7. vuex (^4.1.0)

Vue.js 应用的状态管理库

开发时依赖项

1. vitejs/plugin-vue (^4.3.4)

Vite 构建工具对 Vue 单文件组件的支持插件

2. autoprefixer (^10.4.16)

PostCSS 插件，用于自动添加 CSS 前缀以确保跨浏览器兼容性

3. postcss (^8.4.30)

用于处理 CSS 的工具，通常与 Autoprefixer 一起使用

4. tailwindcss (^3.3.3)

一个实用的 CSS 框架，用于快速构建现代化的用户界面

5. vite (^4.4.9)

下一代前端构建工具，用于快速开发现代化的 Web 应用

4.2.2 部署方式

1. 启动前端服务器：拉取仓库的 master 分支以后运行以下命令

```
1 cd frontend
2 npm install
3 npm run dev
```

2. 启动 nginx 转发服务：配置并且运行 nginx 服务器，进入 nginx 的 conf 文件并修改具体内容如下：

```
1 listen 80;
2 server_name localhost;
3
4 location / {
5     # 配置前端地址
6     proxy_pass http://localhost:5173;
7     proxy_http_version 1.1;
8     proxy_set_header Upgrade $http_upgrade;
9     proxy_set_header Connection 'upgrade';
10    proxy_set_header Host $host;
11    proxy_cache_bypass $http_upgrade;
12
13    # 解决跨域问题
```

```

14     add_header 'Access-Control-Allow-Origin' '*';
15     add_header 'Access-Control-Allow-Methods' 'GET, POST, OPTIONS';
16         add_header    'Access-Control-Allow-Headers'    'DNT,Web-
17 MSSignature,accept,origin,content-type';
18     if ($request_method = 'OPTIONS') {
19         return 204;
20     }
21 }
22
23 location /api/ {
24     # 配置后端地址
25     proxy_pass http://localhost:8080;
26     proxy_http_version 1.1;
27     proxy_set_header Upgrade $http_upgrade;
28     proxy_set_header Connection 'upgrade';
29     proxy_set_header Host $host;
30     proxy_cache_bypass $http_upgrade;
31
32     # 解决跨域问题
33     add_header 'Access-Control-Allow-Origin' '*';
34     add_header 'Access-Control-Allow-Methods' 'GET, POST, OPTIONS';
35         add_header    'Access-Control-Allow-Headers'    'DNT,Web-
36 MSSignature,accept,origin,content-type';
37     if ($request_method = 'OPTIONS') {
38         return 204;
39     }
40 }
41
42 # 可以添加其他配置, 如日志等
43
44 # 如果需要 HTTPS, 可以添加 SSL 配置
45 # ssl_certificate /path/to/your/certificate.crt;
46 # ssl_certificate_key /path/to/your/private/key.key;
47 # ssl_protocols TLSv1.2 TLSv1.3;
48 # ssl_ciphers 'TLS_AES_128_GCM_SHA256:TLS_AES_256_GCM_SHA384';
49 # ssl_prefer_server_ciphers off;
50 }
```

完成上面 2 步以后刷新浏览器，如果出现以下画面说明配置成功，如果没有内容产生，f12 打开控制台，如果出现 cros 报错说明 nginx 配置存在问题，请仔细检查。

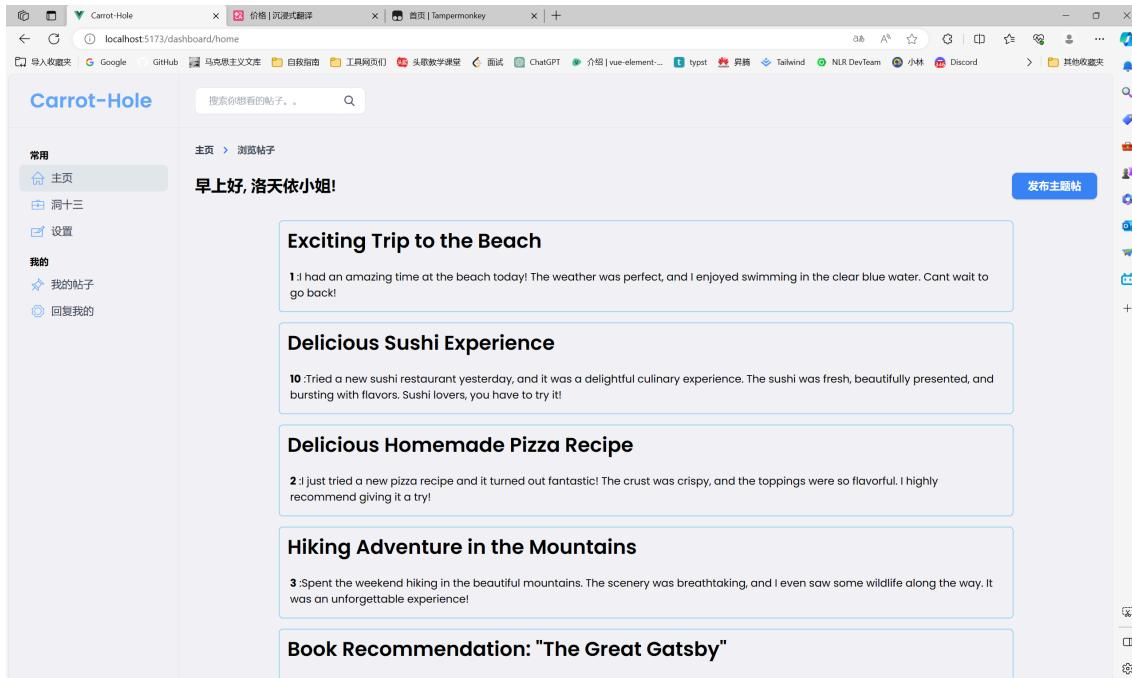


图 18: 正确显示

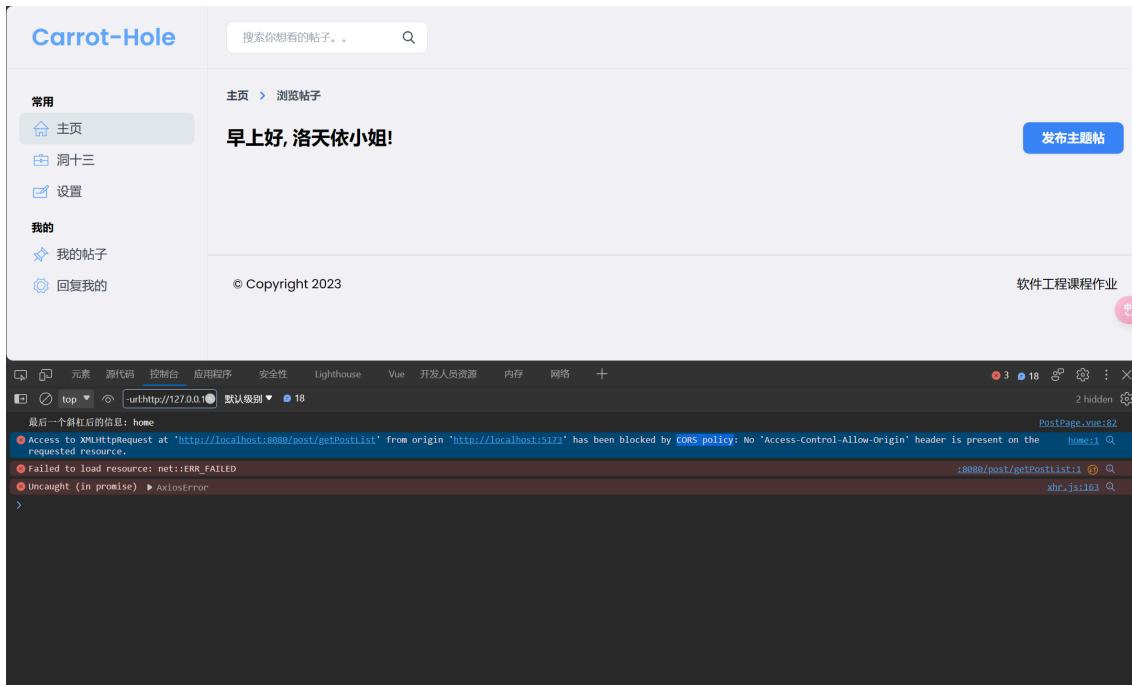


图 19: 错误显示

如果没有正确显示，如上 f12 打开浏览器控制台显示 CROS，说明不显示的原因是 nginx 配置错误。

5 实现与测试

5.1 前后端通信

项目使用 axios 来实现服务器和客户端之间的通信。Axios 是一个基于 Promise 的 HTTP 客户端，用于浏览器和 Node.js 环境。它是一个流行第三方库，提供了简单而强大的用于进行 HTTP 请求的接口。Axios 支持异步操作，能够处理请求和响应的转换，还提供了拦截器（interceptors）等功能，使得在处理 HTTP 请求时更加方便和灵活。

5.1.1 后端 API 测试

下表是所有可访问的 API。

接口名称	请求方法	请求路径
注册	post	/auth/register
登录	post	/auth/login
发表帖子	post	/post/publish
获取帖子	post	/post/get
搜索帖子	post	/post/search
获取主页帖子列表	get	/post/getPostList
获取用户帖子列表	post	/post/getMyPost
发表评论	post	/comment/publish
获取帖子评论	post	/comment/getPostComment

接下来使用 postman 对 API 进行测试。Postman 是一款流行的 API（应用程序编程接口）开发工具，用于测试、调试和发布 Web 服务。它提供了一个用户友好的界面，使开发人员能够轻松地创建、调试和测试 API 请求，同时支持自动化测试和生成文档。下列是每个 API 的详细请求数据和返回数据以及请求的测试结果，通过结果可知，正确的请求能够获取正确的返回结果。

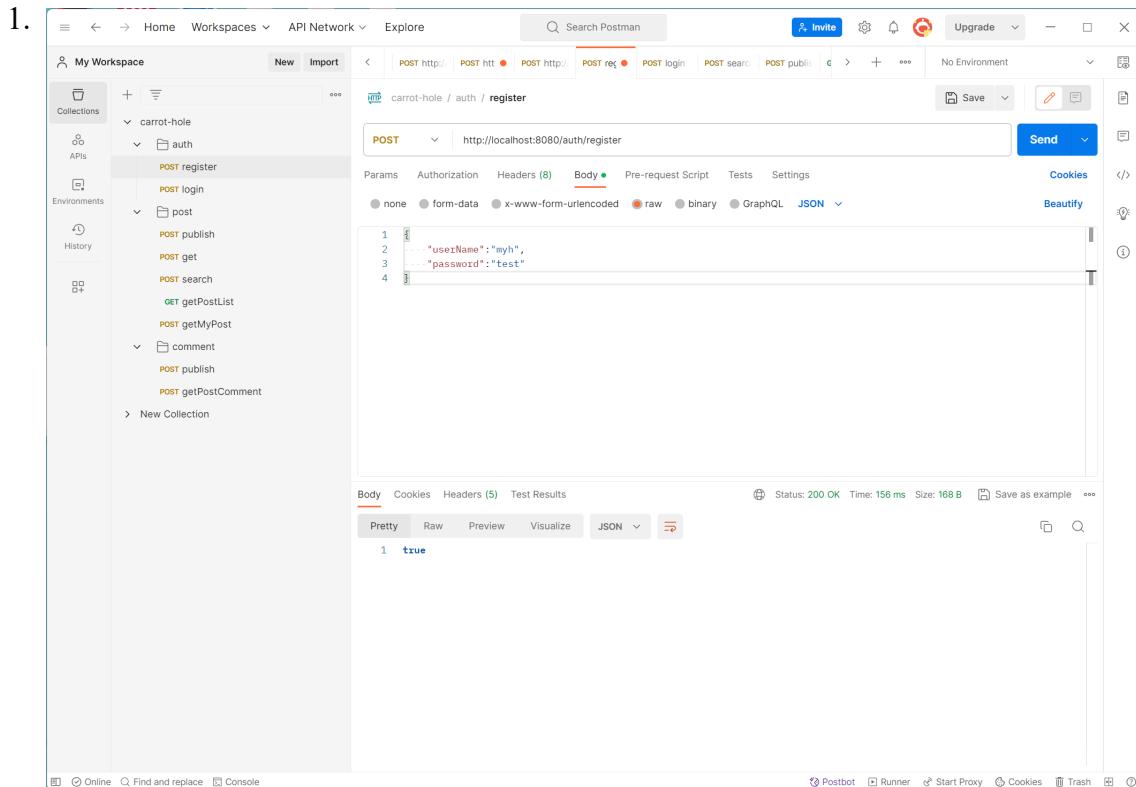


图 20: 注册

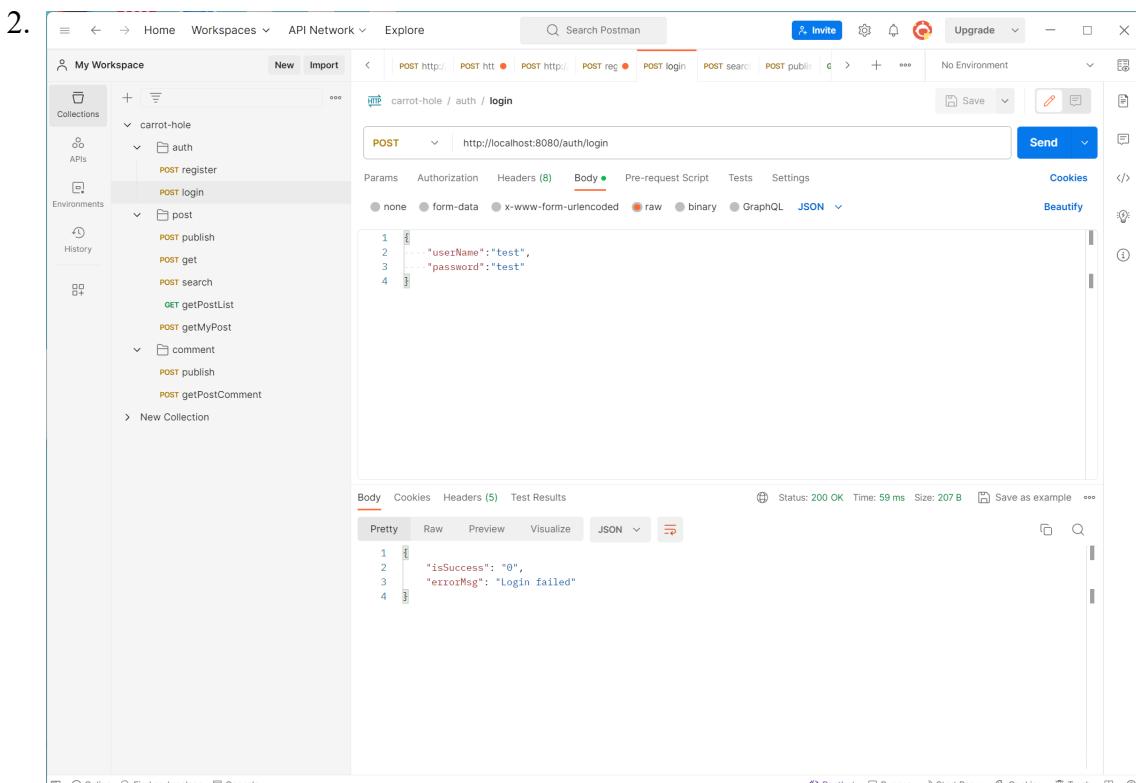


图 21: 登录

3.

The screenshot shows the Postman application interface. In the left sidebar, there's a collection named 'carrot-hole' containing several API endpoints under 'post'. One endpoint, 'POST publish', is selected. The main area shows a POST request to 'http://localhost:8080/post/publish'. The 'Body' tab is selected, showing a JSON payload:

```

1 ...
2 ...
3 ...
4 ...
5 ...
  
```

The payload contains three fields: 'postTitle', 'postContent', and 'userId'. Below the request, there's a 'Response' section with a small cartoon illustration of an astronaut launching a rocket.

图 22: 发表帖子

4.

This screenshot is identical to Figure 22, showing the same POST request to 'http://localhost:8080/post/publish' with the same JSON payload. The interface includes the same collection structure and the same JSON payload in the 'Body' tab.

图 23: 获取帖子

5.

The screenshot shows the Postman interface with a collection named "carrot-hole". A POST request is selected with the URL `http://localhost:8080/post/publish`. The "Body" tab is active, showing JSON input:

```

1  ...
2  ...
3  ...
4  ...
5
  
```

The response area contains a cartoon illustration of an astronaut launching a rocket.

图 24: 搜索帖子

6.

The screenshot shows the Postman interface with a collection named "carrot-hole". A GET request is selected with the URL `http://localhost:8080/post/getPostList`. The "Headers" tab is active, showing a table with one row:

Key	Value	Description
Key	Value	Description

The response area shows the JSON result of the API call:

```

1  "postList": [
2    {
3      "postId": "1",
4      "postTitle": "Exciting Trip to the Beach",
5      "postContent": "I had an amazing time at the beach today! The weather was perfect, and I enjoyed",
6      "postDate": "2023-01-05T00:00:00",
7      "postLike": 10,
8      "userId": "1"
9    },
10   {
11     "postId": "10",
12     "postTitle": "Delicious Sushi Experience",
13     "postContent": "Tried a new sushi restaurant yesterday, and it was a delightful culinary experience. The"
14   }
  
```

图 25: 获取主页帖子列表

7.

The screenshot shows the Postman interface with the following details:

- Collection:** My Workspace > carrot-hole > post > getMyPost
- Method:** POST
- URL:** http://localhost:8080/post/getMyPost
- Headers:** (8)
- Body:** (Pretty) JSON response showing a list of posts:


```

1   "postList": [
2     {
3       "postId": "1",
4       "postTitle": "Exciting Trip to the Beach",
5       "postContent": "I had an amazing time at the beach today! The weather was perfect, and I enjoyed
6           swimming in the clear blue water. Cant wait to go back!",
7       "postDate": "2023-01-05T00:00:00",
8       "postLike": 10,
9       "userId": "1"
10    ],
11   "successMsg": "Success",
12   "isSuccess": "1"
13 ]
14 
```
- Status:** 200 OK, Time: 63 ms, Size: 486 B

图 26: 获取用户帖子列表

8.

The screenshot shows the Postman interface with the following details:

- Collection:** My Workspace > carrot-hole > comment > publish
- Method:** POST
- URL:** http://localhost:8080/comment/publish
- Headers:** (8)
- Body:** (Pretty) JSON response showing a success message:


```

1   "successMsg": "publish success",
2   "isSuccess": "1"
3 
```
- Status:** 200 OK, Time: 96 ms, Size: 212 B

图 27: 发表评论

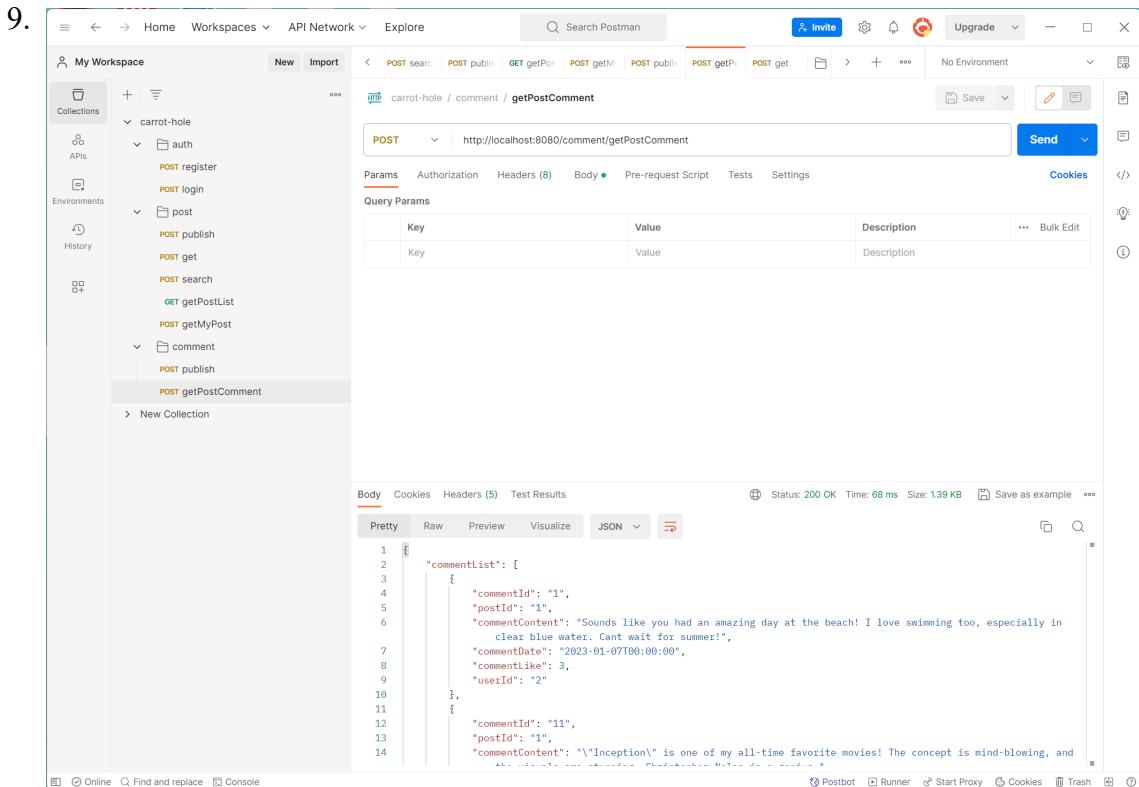


图 28: 获取帖子评论

5.1.2 前端 API 测试

前端使用 axios 库构建 http 请求，在 dev 环境下构建测试页面 <http://localhost:5173/post/test> 测试请求的正确性。项目/frontend/src/utils 文件夹下存储了所需的 http 请求，包括 post,request 等。以下是前端 API 代码

```

1 import axios from 'axios';
2
3 function myAxios(axiosConfig) {
4   const service = axios.create({
5     baseURL: 'http://localhost:8080', // 设置统一的请求前缀
6     timeout: 10000, // 设置统一的超时时间
7   });
8
9   return service(axiosConfig)
10 }
11
12 export default myAxios;

```

基本 axios 对象构建，其他 axios 都将会继承该方法。

```

1 import axios from "axios";
2
3 const postPublishUrl = "http://localhost:8080/post/publish";
4 const commentPublishUrl = "http://localhost:8080/comment/publish";

```

```
5
6 const headers = {
7   "Content-Type": "application/json",
8 };
9
10 export function postPublishAPI(postTitle, postContent, userId) {
11   return axios.post(
12     postPublishUrl,
13     {
14       postTitle: postTitle,
15       postContent: postContent,
16       userId: userId,
17     },
18     { headers }
19   );
20 }
21
22 export function commentPublishAPI(postId, userId, commentContent) {
23   return axios.post(
24     commentPublishUrl,
25     {
26       postId: postId,
27       userId: userId,
28       commentContent: commentContent,
29     },
30     { headers }
31   );
32 }
33
34 export function registerAPI(postTitle, postContent, userId) {
35   return axios.post(
36     postPublishUrl,
37     {
38       postTitle: postTitle,
39       postContent: postContent,
40       userId: userId,
41     },
42     { headers }
43   );
44 }
45 export function loginAPI(postTitle, postContent, userId) {
46   return axios.post(
47     postPublishUrl,
48     {
49       postTitle: postTitle,
50       postContent: postContent,
51       userId: userId,
52     },
53     { headers }
54   );
55 }
56 export function publishPostAPI(postId, userId, commentContent) {
57   return axios.post(
```

```

58     commentPublishUrl,
59     {
60       postId: postId,
61       userId: userId,
62       commentContent: commentContent,
63     },
64     { headers }
65   );
66 }
67 export function searchPostAPI(postId, userId, commentContent) {
68   return axios.post(
69     commentPublishUrl,
70     {
71       postId: postId,
72       userId: userId,
73       commentContent: commentContent,
74     },
75     { headers }
76   );
77 }
78 export default { postPublishAPI, commentPublishAPI };

```

```

1 import axios from "axios";
2 const getPostByIdAPIUrl = "http://127.0.0.1:8080/post/get";
3 const getPostListAPIUrl = "http://localhost:8080/post/getPostList";
4 const getPostCommentListAPIUrl =
5   "http://localhost:8080/comment/getPostComment";
6 const headers = {
7   "Content-Type": "application/json",
8 };
9
10 export function getPostByIdAPI(postId) {
11   return axios.post(
12     getPostByIdAPIUrl,
13     {
14       postId: postId,
15     },
16     { headers }
17   );
18 }
19
20 export function getPostListAPI() {
21   return axios.get(getPostListAPIUrl, { headers });
22 }
23
24 export function getPostCommentListAPI(postId) {
25   return axios.post(
26     getPostCommentListAPIUrl,
27     {
28       postId: postId,
29     },
30     { headers }

```

```

31   );
32 }
33 export default {
34   getPostByIdAPI,
35   getPostListAPI,
36   getPostCommentListAPI,
37 };

```

以上代码就是前端 API 请求的全部 JS 代码

值得注意的是，因为 axios 是异步请求，它的返回值是一个 promise 对象。

promise 对象意味着它可以在未来的某个时间点完成，切换状态（仅此一次）并且返回操作的结果（promise 一共有三种状态， pending 进行中， fulfilled 已完成， rejected 已失败）。

因为 promise 对象一定会在获取结果以后状态发生改变，因此可以在状态发生改变时调用函数读取状态并进行相应处理来实现 http 请求中回调函数的功能（因为发出请求需要等待一段时间才能获得返回值）。

综上在项目里使用上述 API 的 http 请求的操作是值得注意的。以 getPostByIdAPI 为例展示使用方法如下。

```

1 getPostByIdAPI(0).then(response => {
2   //let res = response.data;
3   //console.log('title:', response.data.post.postTitle);
4   //console.log('content:', response.data.post.postContent);
5   this.postData = response.data.post;
6   //let postData = response.data.post;
7   //console.log('响应数据：', postData);
8 })
9 .catch(error => {
10   console.error('请求失败：', error);
11 });
12 getPostListAPI().then(response => {
13   //console.log('响应数据：', response.data);
14   //console.log('第一个的 title:', response.data.postList[0].postTitle);
15   this.postList = response.data.postList;
16 })

```

通过代码可知处理 http 请求时使用 then() 函数获取异步返回使用 catch() 函数捕获异常返回，其他 API 请求类似。

5.2 前端路由

Vue 前端路由是一种在 Vue.js 应用中用于导航不同页面（视图）而无需重新加载整个页面的技术。这是实现单页应用（Single Page Application，简称 SPA）的关键技术之一。在传统的多页应用（MPA）中，每次页面跳转都需要从服务器加载新的页面，这会导致页面闪烁和延迟。而在 SPA 中，所有页面的内容都是在单个页面加载后通过前端路由动态更新的，这提供了更流畅和快速的用户体验。

我们运用前端路由的方式是：根据路由导向不同的组件，从而实现页面中部分组件的刷新，而非重载整个网页。实现代码及分析如下：

```

1 import { createRouter, createWebHistory } from "vue-router";
2 const routerPath = [
3   { path: '/', redirect: { name: 'Home' } },
4   { path: '/dashboard', component: HomePage, children: [
5     { path: '/', redirect: { name: 'Home' } },
6     { path: 'home', name: 'Home', component: Home }
7   ],
8 },
9   { path: '/Post', component: HomePage, children: [
10    { path: '/', redirect: { name: 'PostPage' } },
11    { path: 'home/:id', name: 'PostPage', component: PostPage},
12    { path: 'test', name: 'test', component: test}
13  ],
14 },
15   { path: '/ReleasePost', component: HomePage, children: [
16    { path: '/', redirect: { name: 'ReleasePost' } },
17    { path: 'home', name: 'ReleasePost', component: ReleasePost }
18  ],
19 },
20   { path: '/LoginPage', component: LoginPage, children: [
21    { path: '/', name: 'LoginPage' }
22  ],
23 },
24 ];
25
26 const router = createRouter({
27   history: createWebHistory(),
28   routes: routerPath,
29 });

```

1. 用 `createRouter()`方法创建路由器实例。
2. 定义路由规则。在 `routerPath` 数组中定义了一系列路由规则。每个对象都是一个路由规则，包括 `path`, `component`, `redirect`, `children` 等属性。
3. 详解路由。
 - 根路由重定向: `{ path: '/', redirect: { name: 'Home' } }` 表示当用户访问应用的根 URL (/) 时，会自动重定向到名为 Home 的路由。

- Dashboard 路由: { path: '/dashboard', component: HomePage, children: [...] } 表示当用户访问 /dashboard 路径时, 将渲染 HomePage 组件。这个路由还定义了子路由: 访问 /dashboard/ 时会重定向到 /dashboard/home, 访问 /dashboard/home 时会渲染 Home 组件。
- Post 路由: { path: '/Post', component: HomePage, children: [...] } 类似于 Dashboard 路由, 这个路由定义了当访问 /Post 时, 应该渲染的组件和子路由。它包括对特定帖子的路由 (/Post/home/:id) 和一个测试页面 (/Post/test)。
- ReleasePost 路由: { path: '/ReleasePost', component: HomePage, children: [...] } 这个路由定义了发布帖子的页面, 当访问 /ReleasePost 时, 将渲染 HomePage 组件和 ReleasePost 组件。
- LoginPage 路由: { path: '/LoginPage', component: LoginPage, children: [...] } 这个路由用于登录页面, 当访问 /LoginPage 时, 将渲染 LoginPage 组件。

4. 创建并配置路由器。

使用 `createRouter` 函数创建了一个路由器实例, 并通过 `createWebHistory` 使用了 HTML5 历史模式。然后, 将之前定义的 `routerPath` 作为路由规则传入。

5.3 前端页面显示

页面主要分为三部分, 主页, 帖子详情页和登录页, Vue 会自动跟踪 JavaScript 状态并在其发生变化时响应式地更新 DOM。因此数据和页面显示是双向绑定的, 如果用户更新了 View, Model 的数据也自动被更新了。以下是页面的显示效果。

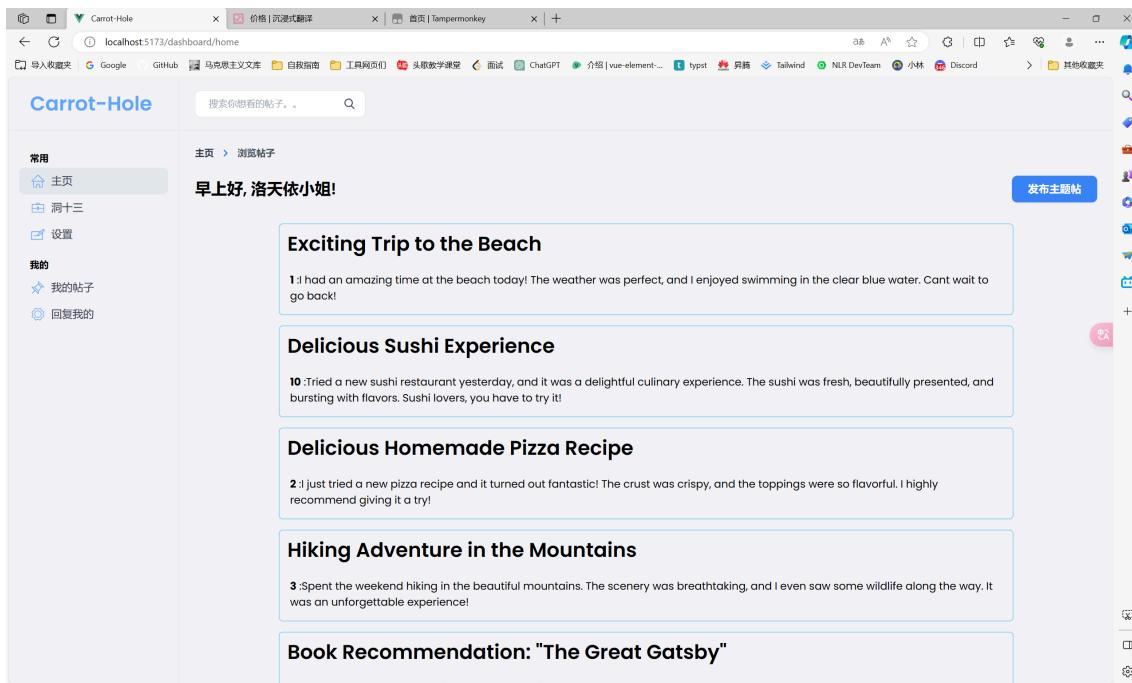


图 29: 主页

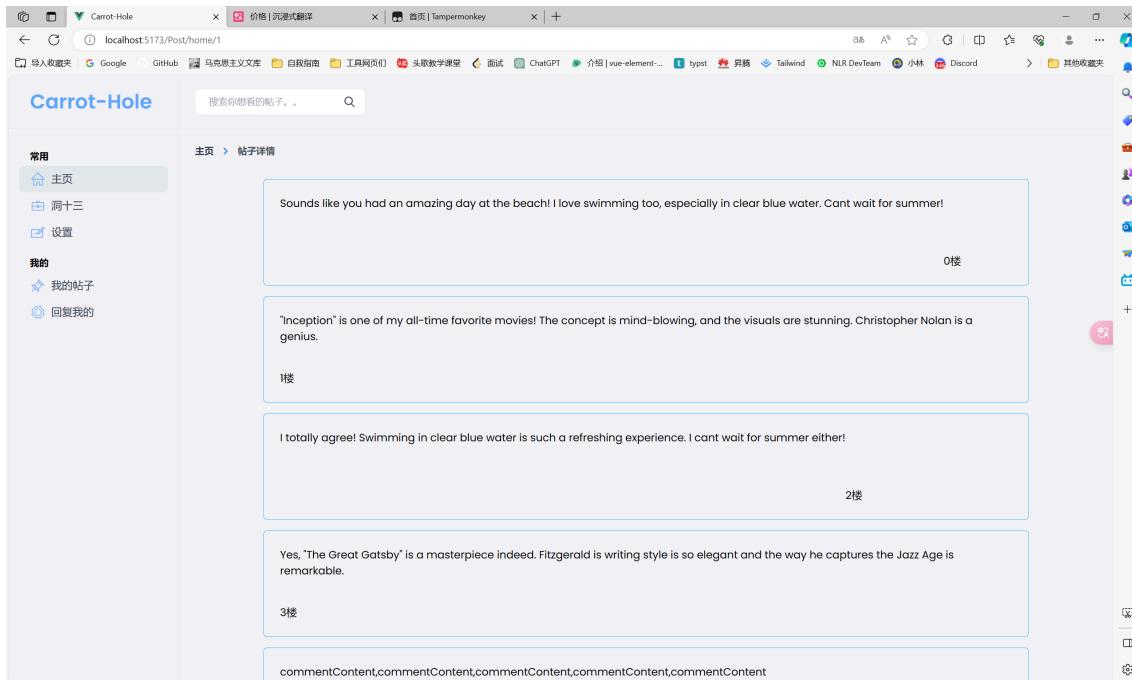


图 30: 帖子详情页

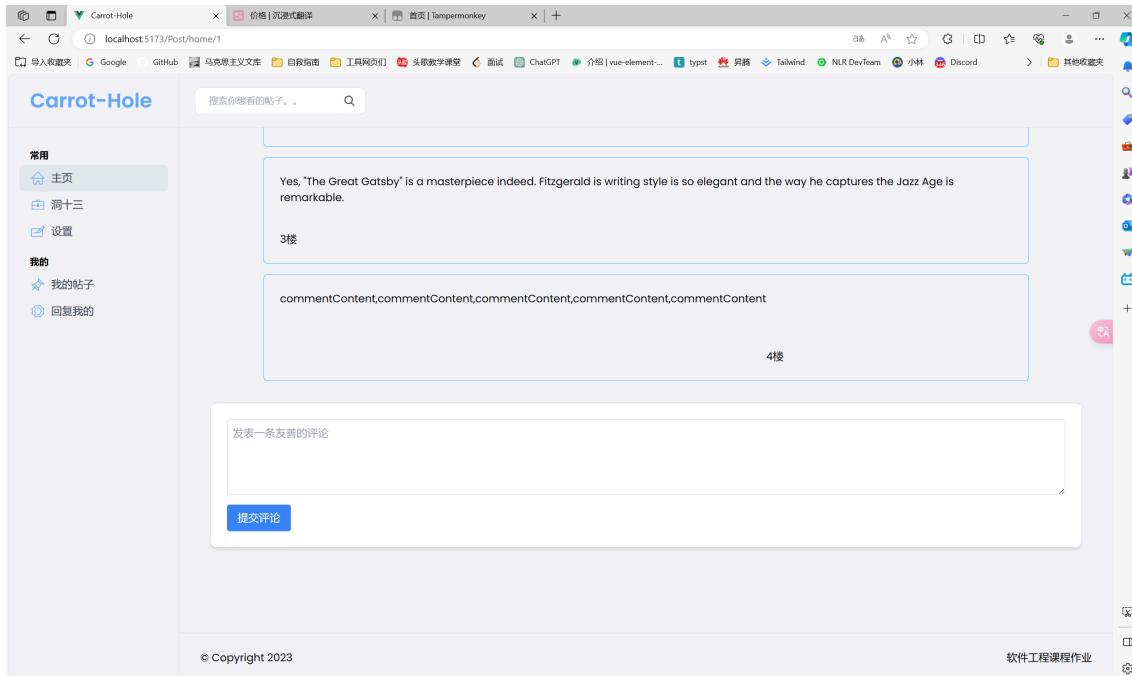


图 31: 登录页

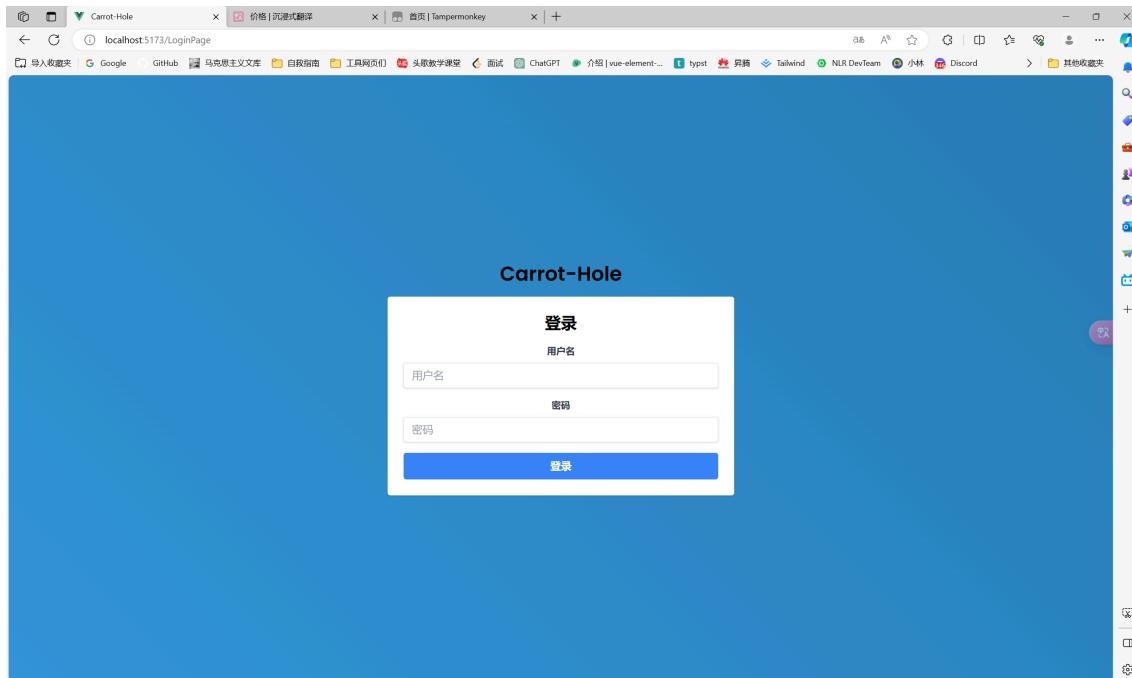


图 32: 发表评论的表单

5.4 结果分析

通过前后端一系列工作实现了诸如浏览帖子，浏览具体帖子每层楼评论，登录，注册，搜索帖子等具体功能，基本实现了一个树洞的要求。

6 总结

6.1 用户反馈

因为不存在真实的用户，因此采取小组内互评的方式。总的来说，我们的树洞项目还存在许多不足，但是也有很多值得称道的地方。

前端的页面不完善，还存在很多没有完成的工作，比如状态转移系统和搜索系统，页面仍较为简陋。但是前端采用了可拓展性很高的先进的架构。VUE 的双向绑定使得 dom 可以动态的更新，nginx 采用反向转发的方式，保证了安全性的同时解决了浏览器的跨域问题，而且具有很高的扩展性。

后端实现了相当完整的系统，实现了诸多的业务逻辑，但也存在许多的瑕疵，例如：没有对数据库中数据进行完整性约束，导致数据存在不统一的可能（修改某张表主键但是另一张表的外键没有做对应的修改）。

6.2 全文总结

在本次课程设计任务中，我们通过合理分工，按照软件设计的基本流程和规范设计了基于前后端分离的论坛 Web App，基本达成了课程设计所提出的要求。

本次工作主要分为需求分析、概要设计、开发与测试等一系列流程。

6.2.1 需求分析

在需求分析阶段，我们首先构建并用文字描述 NABCD 模型；接着，我们绘制了 UML 需求分析图和数据流图，明确了要设计的需求和可能的数据存储模型；最后，我们设计了原型系统。

6.2.2 概要设计

在概要设计阶段，我们分别从环境搭建、工程配置、系统结构、接口声明、服务实现等多个方面阐述了我们的系统设计。

6.2.3 开发与测试

在开发阶段，我们先并行地开发前后端，等基本开发完成后，再整合前后端并对前后端功能分别测试。在测试阶段，我们使用了 PostMan 等请求测试工具，分别对于前后端通信、前端路由、前端界面显示、后端数据库测试等进行了测试。

7 体会

张钧玮：我负责前端的交互设计开发，确保前端的请求能够得到后端的正确响应，并且数据能够顺畅地流动。我通过这个过程掌握了 API 的设计和优化，运用了自己对于网络请求和异步编程的深入理解，确保了用户界面的交互和后端服务的稳定性。

马耀辉：我负责后端设计和开发任务，是项目中极其关键的一个部分，包括了数据库设计、服务器逻辑处理以及确保数据安全性和稳定性。通过这次项目，我深入实践了 Spring Boot 框架，确保了我们的论坛能够处理大量并发请求，同时维护了系统的高性能和安全性。

夏彦文：我承担了前端网页设计和开发的职责。这个角色让我深入了解了前端框架，特别是 Vue.js。在这之前，我已经熟悉了使用 HTML 和 CSS 来搭建基础网页，并且有过 TypeScript + React 的编程经验。通过这次项目，我首次系统地学习并实践了 Vue.js，在构建动态和响应式的用户界面方面取得了实质性的进展。

总而言之，我们通过这个项目深刻理解了软件开发的多面性和团队协作的重要性。

8 附录

附录源代码见 `../carrot-hole`