[redacted]

[redacted]说明书

题 目 基于 Django 的网络相册设计与实现

[redacted]

二 〇 二 三 年 五 月 二 十 六 日

[redacted]

摘　　要

随着互联网快速发展以及 WEB3.0 的普及，网络相册已经逐渐成为人们分享、储存照片的一种新兴方式。本文旨在基于 Djano 实现一个前后端分离的网络相册应用。

本文明确了网络相册设计的目标和需求。考虑到用户的需求和体验，以数据安全为基础，以用户体验为中心，关注相册的功能性和用户界面设计，旨在提供一个简洁、易用且具有良好扩展性的相册平台。

本文采用了系统化的方法进行设计和实现。首先，对相册平台的需求进行了分析和规划，包括用户管理、相片上传、相片展示和社交互动等功能。然后，基于 Django 框架和相关的 Web 开发技术，构建了相册平台的核心功能和模块。在设计过程中，充分考虑了系统的可扩展性、性能和安全性等因素，并采用了合适的数据存储和访问策略。

关键词：网络相册；Django；前后端分离；WEB3.0

Abstract

With the rapid development of Internet and the popularity of WEB3.0, web album has gradually become an emerging way for people to share and store photos. The purpose of this paper is to implement a web album application based on Djano with separated front and back ends.

This paper specifies the goals and requirements of the web album design. Considering the users' needs and experiences, we focus on the functionality and user interface design of the album based on data security and user experience, and aim to provide a simple, easy-to-use and well-expanded album platform.

This paper adopts a systematic approach to design and implementation. First, the requirements of the album platform are analyzed and planned, including user management, photo uploading, photo display and social interaction functions. Then, the core functions and modules of the album platform were built based on the Django framework and related web development technologies. During the design process, factors such as system scalability, performance and security were fully considered, and appropriate data storage and access strategies were adopted.

Keywords： web album; Django; front and back-end separation; WEB3.0

目 录

1 绪 论

1.1 选题背景和意义

相册这个概念很早便有，但是早期的相册是以实体的形式存在，随着时代的发展，科学技术水平的日益提高，传统相册已经不足以应对这个信息化和旅游业快速发展的社会[1]。人们拍摄的照片越来越多，如何存储和管理拍摄的照片成为很多人的困扰。近年来，随着 WEB 3.0 概念的广泛普及，人们的社交需求大大增强，开始将相册搬到了数字化的平台上。相比于传统相册，网络相册拥有着更多的优势。首先，网络相册可以实现无限制的存储空间，不受地域和物理空间的限制。其次，网络相册可以通过云存储实现多端同步，让用户可以随时随地访问自己的照片。

网络相册用户可以将上传的照片设置为“私密”或者“公开”属性。当照片状态为“私密”时，此时照片仅能够自己浏览查看；当照片属性为“公开”时，照片可以被网络相册平台的所有用户查看，用户可以点赞，评论，分享图片。网络相册的存在为广大网络用户提供了一个巨大的图片库，用户可以在其中上传自己创作的图片或自己喜欢的图片。

存储设备价格已经较为普通，但是外置设备不足以满足以手机为核心的移动互联网的快速发展。手机的存储空间一直是有限的，随着时间的推移，越来越多的东西需要存储在手机中，这其中很大一部分是用户的照片，随着网络相册的普及，相片上传到云端进行存储，在需要时下载使用，就会节省很大的存储空间。

在这个互联网时代，人们越来越重视数据的安全和隐私保护。因此，本项目将注重用户数据的保护和隐私安全。同时，将建立一个社区模式，让用户可以相互分享和交流他们的照片。为了使用户能够更好地管理和分享他们的照片，将开发一套易用的用户界面和功能。用户可以上传、编辑、整理、分享和存储他们的照片[3]。

最终目标是建立一个安全、易用的网络相册平台，为用户提供一个更好的照片管理和分享体验，同时也为整个互联网社区带来更多的价值和创新。

1.2 国内外研究现状

目前国内外已经有很多公司实现网络相册，如 Google Photos、Flickr、一刻相册、小红书等。

1.2.1 国外研究现状

Google Photos 是由谷歌开发的一款免费的照片管理和分享应用程序，用户可以将照片和视频上传到该应用中，进行整理、编辑、共享和备份。该应用程序采用了智能算法来自动分类、整理和搜索照片，并且可以根据用户的需求生成相册、故事、动画和电影等内容。此外，用户还可以使用 Google Photos 中的高级编辑工具对照片进行修饰、裁剪

和滤镜等操作。

Flickr 是雅虎旗下的一个图片和视频的社交媒体平台，是一个主要面向摄影师和摄影爱好者的网络相册。Flickr 成立于 2004 年，它最初的目标是为用户提供一个在线共享照片和管理照片的平台。Flickr 的特点是注重照片质量和社交互动。它允许用户上传、浏览和分享高质量的照片，同时也提供了社交化的功能，比如添加好友、留言、评论和分享等。此外，Flickr 还支持用户创建和加入兴趣小组，用户可以在小组内分享和交流有共同兴趣的照片和话题。

1.2.2 国内研究现状

一刻相册是百度网盘团队出品的云相册 APP，你可以无限量以原画质备份照片与视频，切换任意手机设备都能访问这些内容。存储在一刻相册云端的照片和视频可以一键下载至你的智能设备中，高速下载，随取随用。一刻相册会根据你拍摄的优质内容自动推荐生成创意照片、卡点视频。你也可以使用像素风格、油画大师功能，为你定制专属像素头像、照片秒变油画风格，让你变身艺术大师。

小红书是一款流行社交应用，用户可以通过发表笔记、短视频、图片等形式分享自己的购物心得、生活经验和心情，也可以浏览其他用户的分享内容，点赞、评论、收藏或分享。

1.3 本文结构

第一章为绪论，写出来网络相册的研究背景以及国内外的研究现状。

第二章为相关技术介绍，对本系统所使用用的相关技术进行和开发框架进行介绍。

第三章为网络相册需求分析，明确了系统开发目标，包括功能需求以及非功能需求。

第四章为网络相册总体设计，明确数据库设计以及功能模块划分。

第五章为网络相册系统实施，对相关技术进行实现，同时实现各模块功能。

第六章为系统测试，对系统进行单元测试和性能测试。

2 相关技术介绍

2.1 RESTful 架构介绍

2000 年，Roy Fielding 在他的博士论文中首次提出 REST 一词，Fielding 将他对互联网上软件的架构原则，定名为 Representational State Transfer（REST）。REST 是一种设计风格，符合这种风格的软件设计则称为 RESTful[4]。

REST 的核心是“资源”，这些“资源”是网络上的实体，如文本，图片，音频，视频等具体的存在。每种资源都可以通过一个统一资源定位符（URI）进行指向，每个资源对应一个特定的 URI。通过访问该资源的 URI，我们可以获取到该资源，因此 URI 就成为了每个资源的地址或唯一标识符。

前后端分离项目离不开后端 API 的设计，RESTful API 是目前相对较为成熟的一套互联网应用 API 设计理论。本项目采用 RESTful API 架构。

2.2 JWT 技术介绍

JWT 全称为 Json Web Token，它是以 JSON 对象的形式在各方之间安全地传输信息。这种信息可以被验证和信任，因为它是经过数字签名的。JWT 由三部分组成，头部、载荷和签名，他们之间使用“.”分割，形如“header.payload.signature”。头部包含 Token 的类型以及加密算法。载荷部分包含对象实体（通常是用户）和其他数据。签名部分由头部和载荷分别进行 Base64 编码之后使用点连接再结合加密算法以及密钥加密形成。

JWT 在前后端分离系统中应用十分广泛，前端向后端发起用户请求时，将 JWT 置于请求头中供后端进行身份验证，认证通过则接受请求并返回数据，否则拒绝请求。它的工作过程如图 2.1 所示。

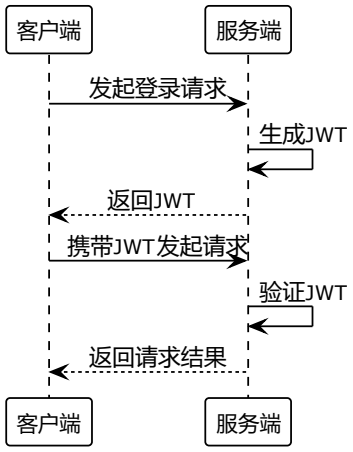

图 2.1 JWT 工作过程

2.3 Django 框架介绍

Django 是一个用 Python 编写的开源 Web 应用程序框架。它采用了 MVC（模型-视图-控制器）的软件设计模式，旨在帮助开发人员快速构建高效且可扩展的 Web 应用程序。Django 框架有如下几个特点：

1. 强大的 ORM（对象关系映射）：Django 提供了一个强大的 ORM 工具，称为 Django ORM，它允许开发人员使用 Python 代码而不是 SQL 语句来定义和操作数据库模型。
2. URL 路由和视图系统：Django 的 URL 路由系统允许开发人员将 URL 映射到特定的视图函数或类。开发人员可以定义 URL 模式，将请求路由到相应的视图

函数，然后在视图函数中处理请求并生成响应。

3. 安全性：Django 框架提供了一系列的安全性功能，包括防止跨站请求伪造（CSRF）攻击、点击劫持保护、XSS（跨站脚本攻击）防护等。

本项目使用基于 Django 框架的扩展 Django REST Framework（DRF）构建 Web API。DRF 框架工作过程如图 2.2 所示。

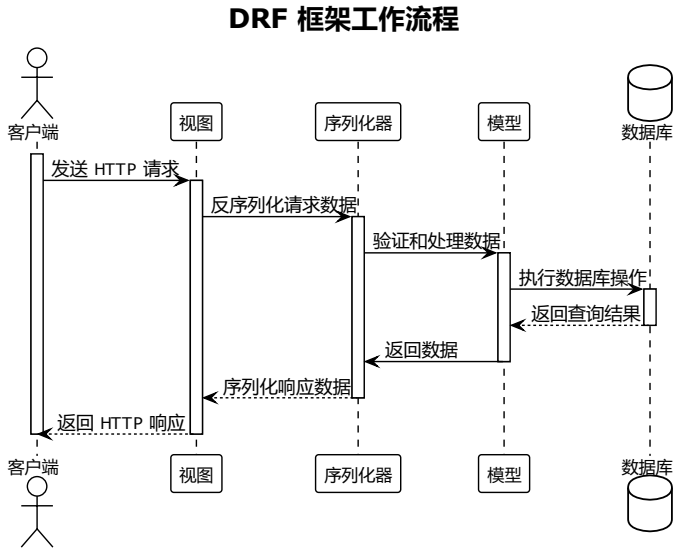

图 2.2 DRF 框架工作流程

2.4 IFPS 介绍

IPFS（InterPlanetary File System）是一个点对点分布式文件系统，旨在创建一个全球性的、内容可寻址的存储和传输网络。它提供了一种新的方式来存储和访问数据，通过将文件分割成小块并使用唯一的哈希标识符来引用它们。IPFS 采用了去中心化的方式，使数据可以在网络中的多个节点上存储和共享，而不是集中在特定的服务器或数据中心。它的核心概念是内容寻址，它使用文件内容的哈希值作为唯一的标识符。只要文件内容相同，它们的哈希值也将相同。这种机制使得在 IPFS 中查找和获取文件变得更加高效和可靠。

2.5 Vue 框架介绍

Vue 是一款用于构建用户界面的 JavaScript 框架。Vue 专注于视图层，采用组件化的方式构建 Web 应用。Vue 使用了响应式的数据绑定机制，当数据发生变化时，视图会自动更新。开发人员只需要关注数据的变化，通过虚拟 DOM 技术，在更新视图时，Vue 会先构建一个虚拟 DOM 树，然后通过比较虚拟 DOM 树和实际 DOM 树的差异，最小化对实际 DOM 的操作，从而提高性能和渲染速度。

Vue 目前最新版本为 3.0，对 TypeScript 有更好支持。本项目前端使用 Vue3 构建，同时使用第三方组件库 Navie UI 和 TypeScript。

3　网络相册需求分析

3.1　网络相册开发目标

本项目所设计的网络相册应具备以下几个方面：

1. 用户友好的界面和操作：网络相册应该提供一个直观、简洁且易于使用的用户界面。用户能够轻松浏览和管理照片，执行常见的操作，如上传、删除、编辑、分享等。
2. 安全和隐私保护：用户的照片和个人信息是敏感的，网络相册应采取安全措施来保护用户的数据。这包括用户身份认证和权限管理，确保其他用户无法访问私密照片。
3. 快速和稳定的性能：网络相册应具备快速响应和稳定的性能，无论是在上传照片、浏览相册还是执行其他操作时，用户都能够获得良好的使用体验。应用的架构和后端系统应能够处理大量的图片数据和用户请求。
4. 多样化的功能和特性：网络相册应提供丰富的功能和特性，以满足用户的需求。这包括相册分类、分区管理、评论和点赞等功能。

3.2　网络相册功能需求分析

3.2.1　游客用户需求分析

对于游客用户，用户只能查看公开图片和相关信息，以及查看分区图片和用户主页，游客用户可以通过注册成为注册用户。游客用户无法上传照片，无法对图片进行点赞、评论等操作。游客用户用例图如图 3.1 所示。

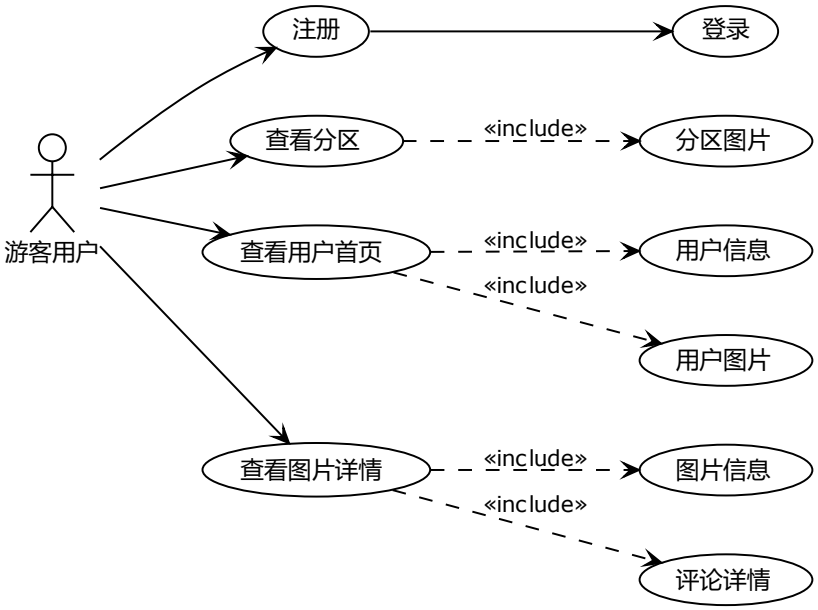

图 3.1 游客用户用例分析

3.2.2 管理员用户需求分析

管理员负责网络相册的后台管理，主要有如下几点需求：

1. 用户管理：管理员可以删除，修改用户，对用户进行冻结。
2. 照片管理：用户上传照片初始状态为待审核，此时管理员可以审核图片。
3. 评论管理：删除违规评论。
4. 分区管理：新增，修改，删除分区。

管理员用例图如图 3.2 所示。

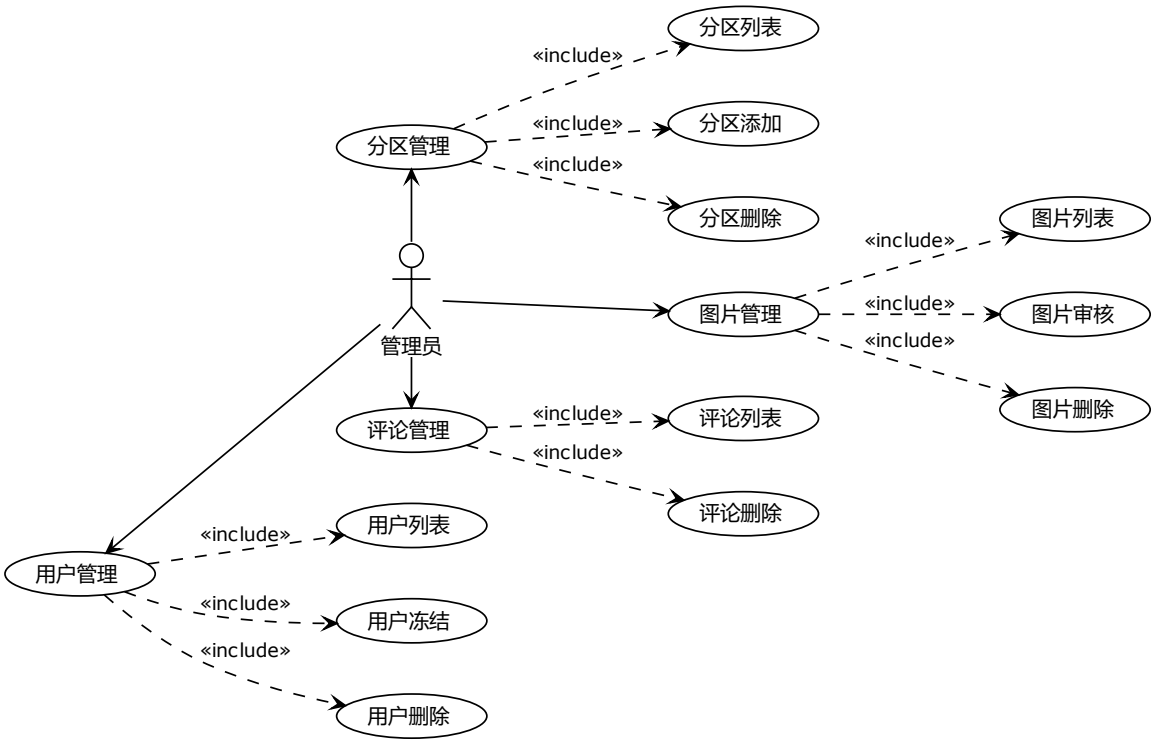

图 3.2 管理员用户用例分析

3.2.3 登录用户需求分析

对于登录用户，主要有如下功能：

1. 上传图片：用户可上传本地图片至网络相册，上传时可以编辑信息，同时选择相册和分区。
2. 个人信息管理：用户修改个人信息，修改密码等。
3. 图片管理：修改图片信息，删除图片，移动相册和分区等
4. 相册管理：创建。修改。删除相册。
5. 其他功能：对图片发表评论，点赞，对评论发表回复。

登录用户用例图如图 3.3 所示。

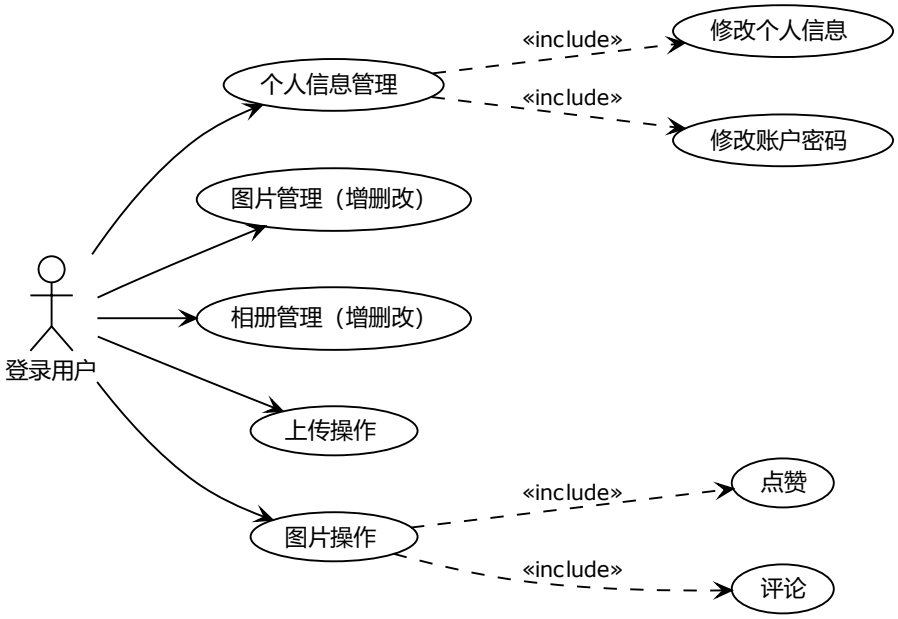

图 3.3 登录用户用例分析

3.3 网络相册非功能需求分析

3.3.1 用户体验需求

用户体验需求是指网络相册应该提供给用户的良好体验和满足用户需求的特性。网络相册的界面应该设计简洁、直观，使用户能够快速上手并且轻松地完成操作。用户应该能够迅速找到所需的功能和选项，而无需过多的学习或搜索。

3.3.2 性能需求

性能需求是指网络相册在处理用户请求和操作时所需的性能特性和要求。网络相册应该具备快速的响应时间，用户发送请求后，系统应该能够迅速作出响应并提供所需的结果。例如，打开相册、浏览照片、上传或下载图片等操作应该在合理的时间范围内完成，以提供流畅的用户体验。同时系统应该能够同时处理多个用户的操作，而不会出现严重的延迟或性能下降。这对于繁忙的相册网站或在高峰期有大量用户访问的情况尤为重要。

3.3.3 安全性需求

安全性需求是指网络相册在保护用户隐私和数据安全方面的特性和要求。网络相册应该提供身份验证机制，确保只有授权用户可以访问和管理照片。同时应该具备日志记录的功能，以便及时检测和应对安全事件。系统应该记录用户的登录和操作行为，以便追踪和分析潜在的安全问题。

4 网络相册总体设计

4.1 系统架构设计

本系统采用前后端分离设计[6]，前端使用 Vue3 框架和 Naive UI 第三方组件库构建，负责实现用户界面和用户交互。Vue3 可以结合 Vue Router 用于路由管理和页面导航，同时使用 Pinia 进行状态管理。使用 Axios 库来处理与后端 API 的 HTTP 通信。

后端采用 Django 框架作为主要的 Web 应用程序框架，并使用 Django REST Framework（DRF）扩展来构建和暴露 API 接口。DRF 提供了强大的工具和功能，用于快速开发和管理 RESTful API.使用 Django 的 ORM 来处理数据持久化和数据库操作。

使用 MySQL 作为数据库，利用 Redis 做缓存，使用 IPFS 作为文件存储。系统架构如图 4.1 所示。

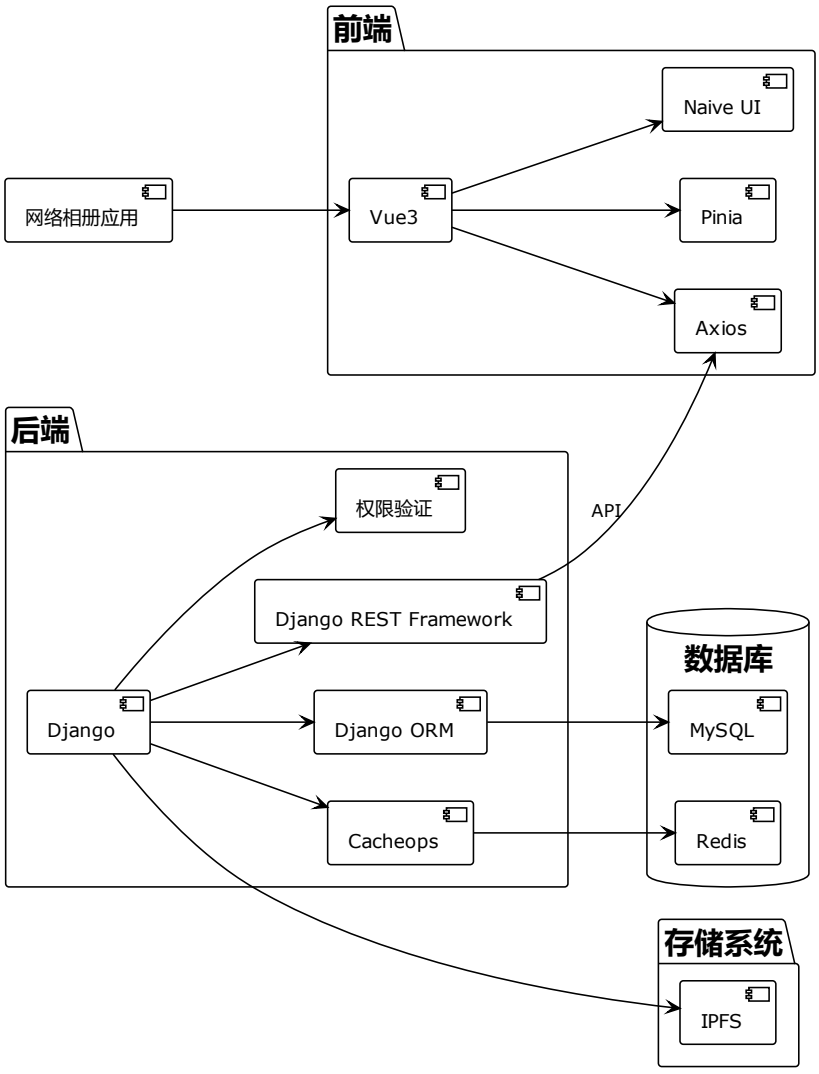

图 4.1 网络相册架构图

4.2 数据库设计

4.2.1 数据库概念设计

数据库概念设计是在建立数据库之前的一个阶段，旨在理解和定义数据库系统的结构、组织和关系。本项目数据库概念设计图如图 4.2 所示。

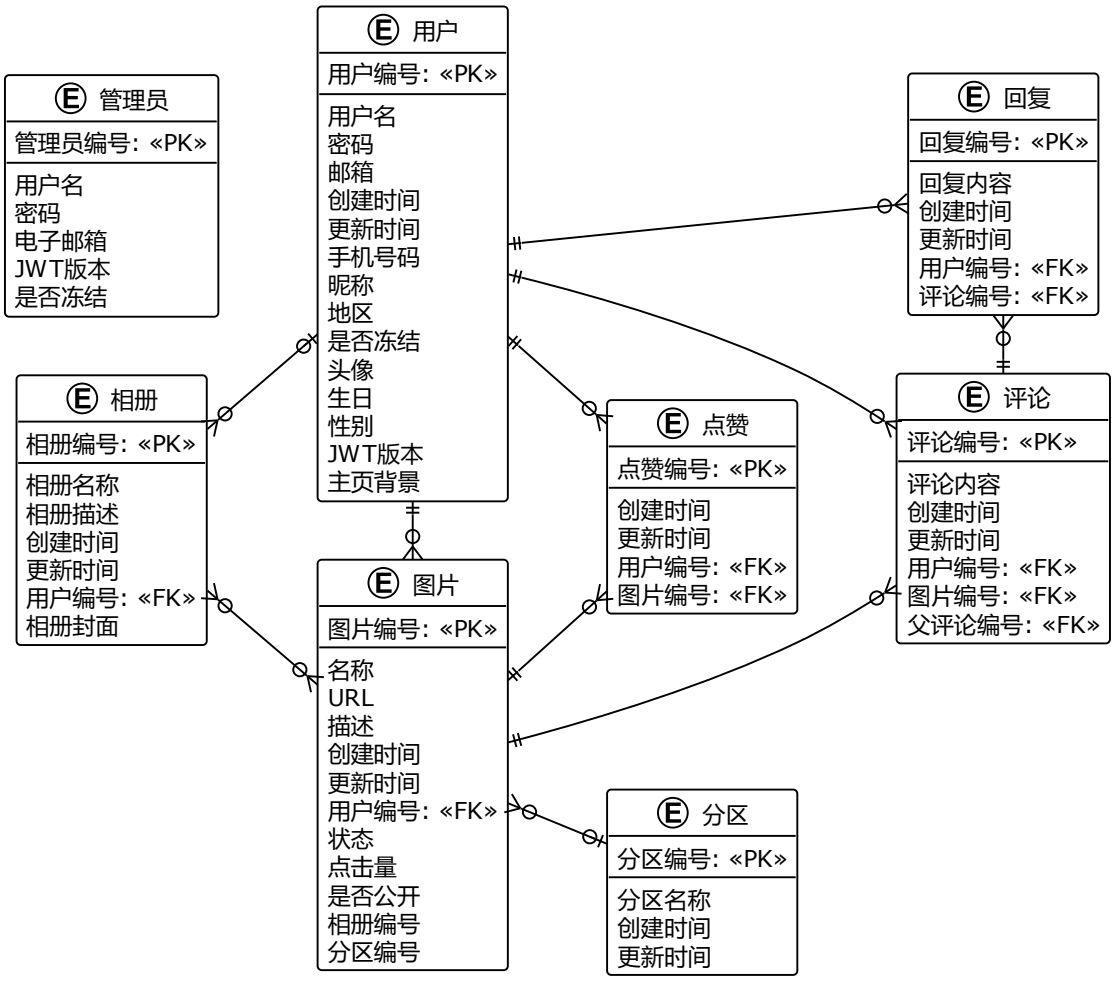

图 4.2 数据库概念设计图

通过数据库的概念模型，可以得到各个实体之间的逻辑关系。例如，用户可以上传图片、管理相册、发表评论等。此外，还可以直观了解到实体具有的属性，这为设计数据库表提供了便利。

4.2.2 数据库表设计

结合网络相册需求和 4.2.1 节中的数据库概念设计图，可以为网络相册设计具体数据表。数据库各表的设计如表 4.1 至表 4.8 所示。

1. 用户表（a_user）

用户表主要用来存储用户的账号、密码、JWT 版本、是否冻结以及其他基本信息。其中 JWT 版本会在用户每一次登陆时自增，避免不同设备同时登录，提高账户安全性。

用户表如表 4.1 所示

表 4.1 用户表（a_user）

序号	字段名称	字段类型	备注
1	id	bigint	用户编号，主键，自增，非空
2	username	varchar(32)	用户名，非空
3	password	varchar(255)	密码（加密），非空
4	email	varchar(255)	邮箱，非空
5	mobile	varchar(11)	手机号码，非空
6	nickname	varchar(32)	昵称，非空
7	regions	varchar(255)	地区，非空
8	avatar	varchar(255)	头像，非空
9	birthday	date	生日，非空
10	cover	varchar(255)	主页背景，非空
11	gender	varchat(1)	性别，0：男性，1：女性，2：未知，非空
12	is_freeze	tinyint(1)	是否冻结，1 表示冻结，0 表示正常，非空
13	jwt_version	int	jwt 版本，初始为 0，非空
14	create_time	bigint	创建时间，Unix 时间戳，非空
15	update_time	bigint	更新时间，Unix 时间戳，非空

2. 图片表（a_photo）

图片表中保存图片的基本信息包括图片编号、图片 URL、图片描述等，其中还有相关外键关联用户、相册、分区等。图片表如表 4.2 所示。

表 4.2 图片表（a_photo）

序号	字段名称	字段类型	备注
1	id	bigint	图片编号，主键，自增，非空
2	name	varchar(32)	图片名，非空
3	url	varchar(255)	图片链接，非空
4	desc	varchar(255)	图片描述，非空
5	click	int	点击量，非空
6	status	int	状态，0：正常，1：待审，2：不通过，非空
7	is_public	tinyint(1)	是否公开，1：公开，0：私密，非空
8	jwt_version	int	jwt 版本，初始为 0，非空
9	album_id	bigint	相册编号，外键
10	partition_id	bigint	分区编号，外键
11	create_time	bigint	创建时间，Unix 时间戳，非空
12	update_time	bigint	更新时间，Unix 时间戳，非空

3. 相册表（a_album）

相册表用来保存相册相关信息，如相册封面，相册描述等，其中有外键用户编号关联用户表。相册表如表 4.3 所示。

表 4.3 相册表（a_album）

序号	字段名称	字段类型	备注
1	id	bigint	相册编号，主键，自增，非空
2	name	varchar(32)	相册名，非空
3	cover	varchar(255)	相册封面
4	desc	varchar(255)	相册描述
5	author_id	bigint	用户编号，外键，非空
6	create_time	bigint	创建时间，Unix 时间戳，非空
7	update_time	bigint	更新时间，Unix 时间戳，非空

4. 评论表（a_comment）

评论表保存图片相关评论信息，其中有外键图片编号，用户编号分别关联图片和用户。评论表如表 4.4 所示。

表 4.4 评论表（a_comment）

序号	字段名称	字段类型	备注
1	id	bigint	评论编号，主键，自增，非空
2	content	varchar(255)	评论内容，非空
3	parent_comment_id	bigint	父评论编号
4	photo_id	bigint	图片编号，外键，非空
5	author_id	bigint	用户编号，外键，非空
6	create_time	bigint	创建时间，Unix 时间戳，非空
7	update_time	bigint	更新时间，Unix 时间戳，非空

5. 回复表（a_reply）

回复表保存评论相关回复内容，其中有外键父评论编号关联评论表。回复表如表 4.5 所示。

表 4.5 回复表（a_reply）

序号	字段名称	字段类型	备注
1	id	bigint	回复编号，主键，自增，非空
2	content	varchar(255)	回复内容，非空
3	parent_comment_id	bigint	评论编号，外键，非空
4	author_id	bigint	用户编号，外键，非空
5	create_time	bigint	创建时间，Unix 时间戳，非空
6	update_time	bigint	更新时间，Unix 时间戳，非空

6. 点赞表（a_like）

点赞表记录用户点赞信息，其中有外键图片编号，用户编号分别关联图片和用户。点赞表如表 4.6 所示。

表 4.6 点赞表（a_like）

序号	字段名称	字段类型	备注
1	id	bigint	点赞编号，主键，自增，非空
3	photo_id	bigint	图片编号，外键，非空
4	author_id	bigint	用户编号，外键，非空
5	create_time	bigint	创建时间，Unix 时间戳，非空
6	update_time	bigint	更新时间，Unix 时间戳，非空

7. 分区表（a_partition）

分区表中记录分区基本信息。分区表如表 4.7 所示。

表 4.7 分区表（a_partition）

序号	字段名称	字段类型	备注
1	id	bigint	分区编号，主键，自增，非空
2	name	varchar(32)	分区名称，非空
3	create_time	bigint	创建时间，Unix 时间戳，非空
4	update_time	bigint	更新时间，Unix 时间戳，非空

8. 管理员表（a_admin）

管理员负责管理整个网络相册，其中账号密码初始存在，不接受外部修改。管理员表如表 4.8 所示。

表 4.1 用户表（a_user）

序号	字段名称	字段类型	备注
1	id	bigint	管理员编号，主键，自增，非空
2	username	varchar(32)	用户名，非空
3	password	varchar(255)	密码（加密），非空
4	email	varchar(255)	邮箱，非空
5	is_freeze	tinyint(1)	是否冻结，1 表示冻结，0 表示正常，非空
6	jwt_version	int	jwt 版本，初始为 0，非空

4.3 系统功能结构设计

根据系统功能分析，网络相册一共可以分为三种用户，分别是游客用户，注册用户和管理员用户。他们的功能结构如图 4.3 所示。

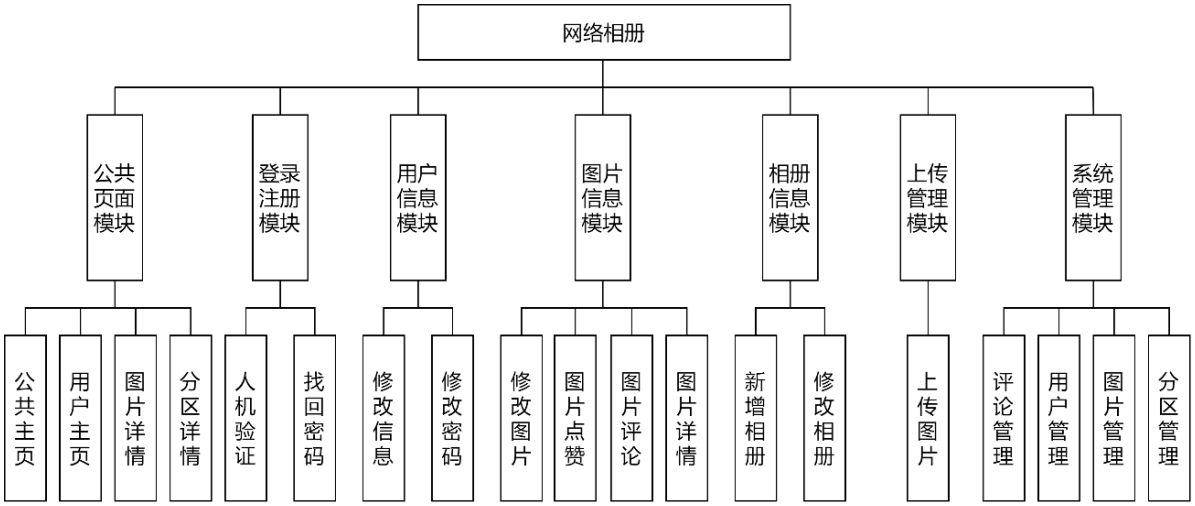

图 4.3 网络相册功能结构图

4.4 系统功能模块说明

1. 公共页面模块

该模块主要服务用户为游客用户，用户无需登录可以使用网络相册部分功能。用户可以查看网络相册主页，主页展示用户上传的公开照片；点击主页图片进入图片详情，图片详情页展示图片基本信息，同时展示图片相关的评论和回复；用户可以进入用户主页查看该用户上传的所有公开照片；同时可以查看某一分区下的所有公开照片。公共页面模块时序图如图 4.4 所示。

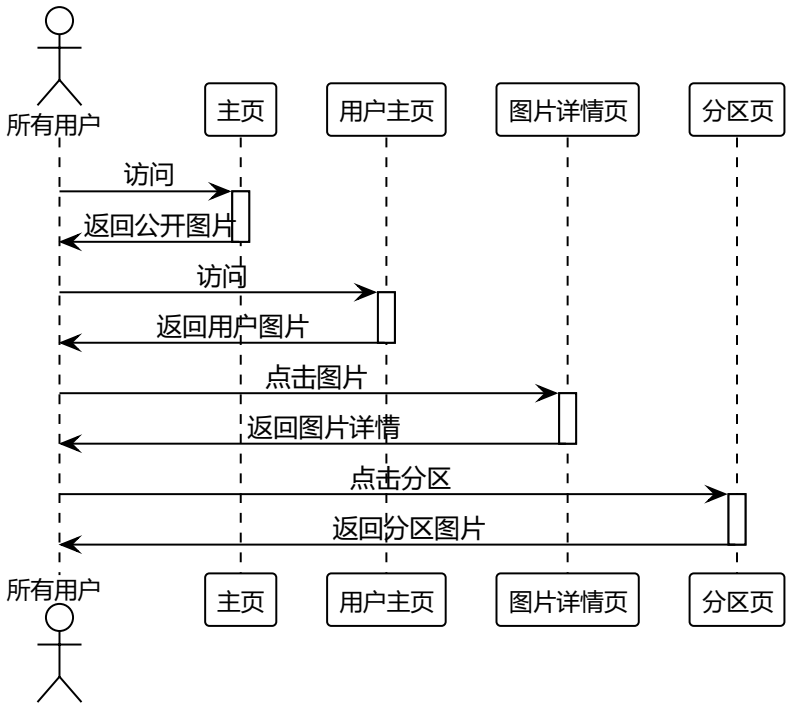

图 4.4 公共页面模块时序图

2. 登录注册模块

游客用户可以通过该模块注册成为注册用户，注册用户可以通过登录成为登录用户。用户在注册或者登录时需要通过人机验证，人机验证使用滑动拼图实现。注册用户若是忘记密码，可以通过找回密码功能找回密码，找回密码通过给用户注册时提供的邮箱发送验证码实现验证。登录注册模块时序图如图 4.5 所示

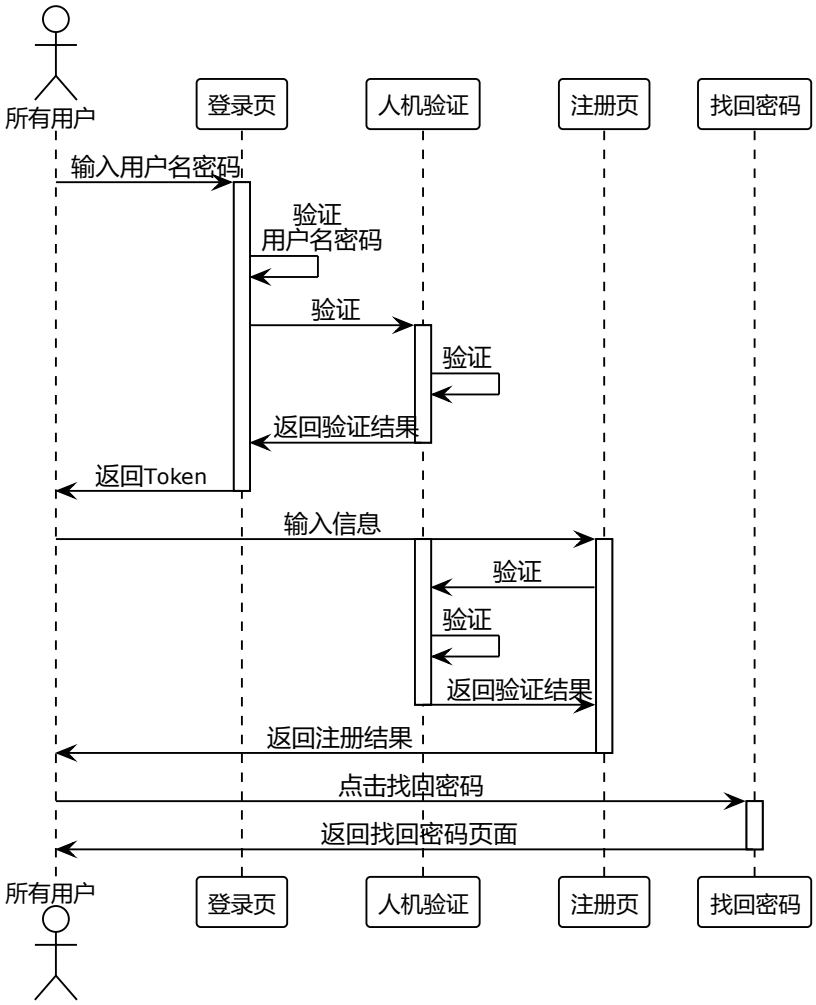

图 4.5 登录注册模块时序图

3. 用户信息模块

该模块主要服务登录用户。用户可以查看个人信息，以及对个人信息进行修改，主要包括修改头像、修改昵称、修改生日、修改地区等。用户可以对个人账号密码进行修改，修改密码与找回密码流程类似，需要用户提供邮箱验证。用户信息模块时序图如图 4.6 所示。

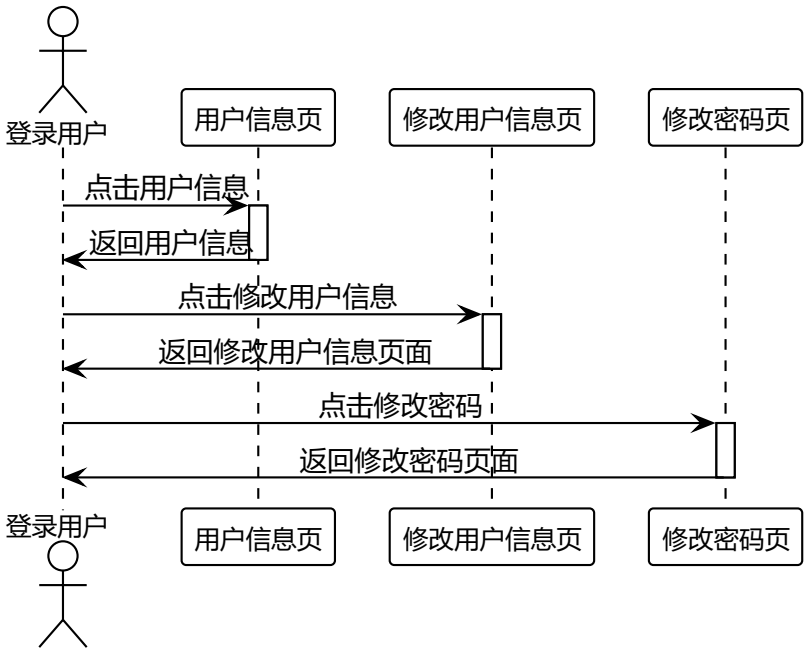

图 4.6 用户信息模块时序图

4. 图片信息模块

登录用户可以对图片进行点赞，点赞数据会保存在数据库中，用户可以查看自己的点赞图片。用户每查看一次图片，图片浏览量会增加一次。用户对图片发表评论，评论功能包含敏感词过滤，若是评论内容包含预设的敏感词，则会被替换成“*”；其他用户可以对评论发表回复，回复功能同样包含敏感词过滤。

用户可以对已上传的图片的基本信息进行修改，包括修改图片描述、修改图片名称、修改分区、修改相册、修改属性等功能；同时可以删除不满意的图片。图片信息模块时序图如图 4.7 所示。

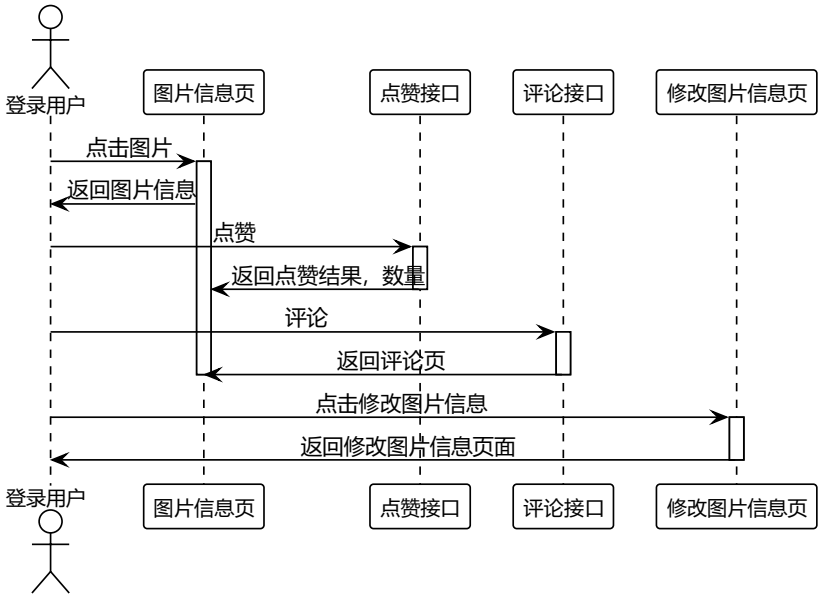

图 14.7 图片信息模块时序图

5. 相册信息模块

相册用途是用来存放同类图片，登录用户可以创建相册，删除相册，其中创建相册可以选择添加相册封面，相册描述；用户在删除相册时，不会删除相册中的图片[7]。相册信息模块时序图如图 4.8 所示。

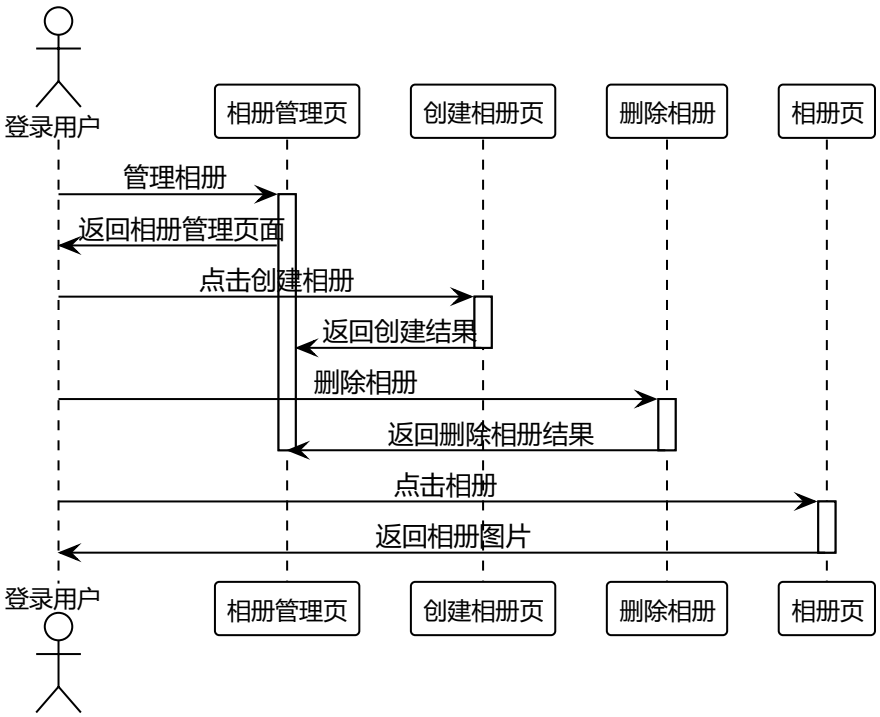

图 4.8 相册信息模块时序图

6. 上传管理模块

登录用户可以上传图片，在图片上传过程中，可以填充图片基本信息，选择相册和分区，上传时要求选择图片是否公开，若图片为公开属性，所有人都可以访问该图片；若图片为私密属性，只能自己访问。上传的图片经过后端系统处理后保存在 IPFS 文件系统中。上传管理模块时序图如图 4.9 所示。

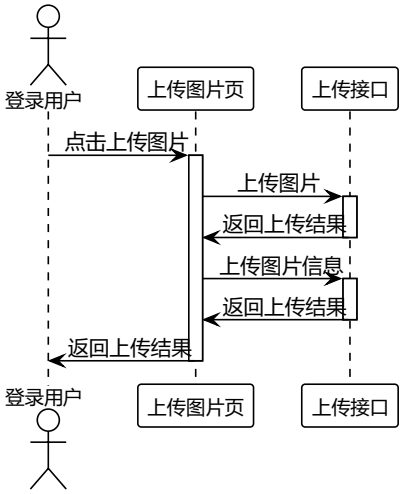

图 4.9 上传管理模块时序图

7. 系统管理模块

该系统独立与网络相册，主要管理整个网络相册。其中包括用户管理，管理员可以选择删除用户或者冻结用户，对于被冻结的用户，用户无法登录网络相册。

图片管理，用户上传的图片初始状态为待审核，管理员可以选择审核照片为正常或者审核不通过，审核照片仅针对公开照片，用户上传的私密照片默认为正常，照片不通过审核则无法展示在网络哟相册中。

评论管理，管理员对用户发表的评论进行管理，对于不符合法律法规的评论，首先有敏感词过滤，再有管理员管理。

分区管理，管理员可以创建、删除、修改分区，分区名称仅由管理员决定，其他用户无法干预。

系统管理模块时序图如图 4.10 所示。

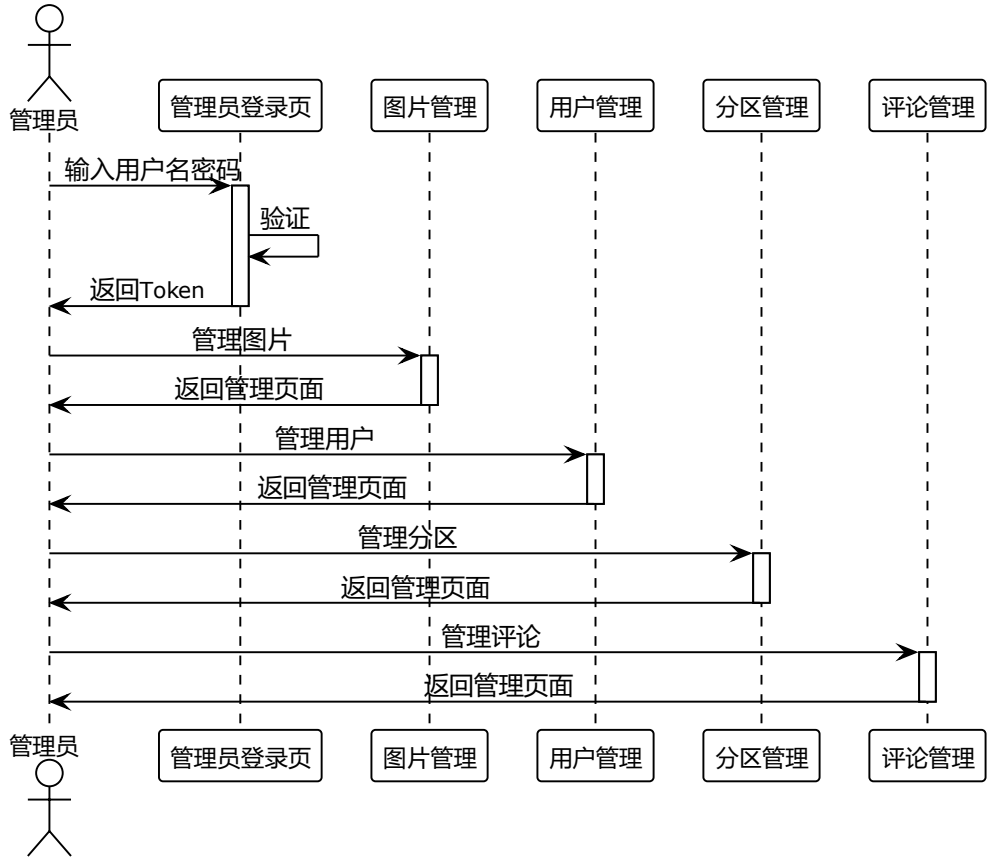

图 4.10 系统管理模块时序图

5 网络相册系统实施

5.1 相关技术的实现

5.1.1 跨域功能实现

跨域资源共享（CORS）是一种机制，它允许 Web 浏览器向跨域服务器发送异步 HTTP 请求，以便获取和使用数据[10]。对于前后端分离的系统，往往前端和后端的端口不同，此时前端发起的请求为跨域请求。在默认情况下，Web 浏览器会限制这种类型的请求，从而保护 Web 应用的安全性。对此，可以通过编写 Django 中间件实现跨域请求。核心代码如下。

```
class CorsMiddleware(MiddlewareMixin):
    def process_request(self, request):
        if request.method == "OPTIONS":
            response = HttpResponse()
        else:
            response = None
        self.set_cors_headers(response)
        return response
    def process_response(self, request, response):
        self.set_cors_headers(response)
        return response
    def set_cors_headers(self, response):
        response["Access-Control-Allow-Origin"] = "*"
        response["Access-Control-Allow-Methods"] = "GET, POST, PUT,
DELETE, OPTIONS"
        response["Access-Control-Allow-Headers"] = "Content-Type,
Authorization"
```

上述代码定义了一个名为 CorsMiddleware 的类，它包含了三个方法：process_request，process_response 和 set_cors_headers。process_request 方法在每个请求处理之前调用，而 process_response 方法在每个请求处理之后调用，set_cors_headers 用以设置请求标头。

在 process_request 方法中，如果请求是 OPTIONS 方法，中间件将返回一个包含 CORS 标头的空 HTTP 响应。这是因为当浏览器执行跨域请求时，它首先会发送一个 OPTIONS 请求，以确定是否可以发送实际的请求。如果服务器返回了包含必要 CORS 标头的响应，则浏览器将发送实际的请求。

在 process_response 方法中，中间件将为每个响应添加 CORS 标头。这将允许来自其他域名的请求访问该响应。

5.1.2 接口缓存功能实现

本项目为前后端分离项目，为了提升用户体验，采用了接口缓存技术。一般来说，对于浏览器发送的请求，后端框架会根据请求中的参数去查询数据库，然后返回响应结果。由于访问数据库存在时延，特别是在请求量高的情况下，时延高会严重影响用户的体验。采用缓存数据库能够有效解决该问题，浏览器第一次请求后端接口时，会访问 Redis 缓存，若缓存命中，则直接返回数据；若缓存未命中，会请求数据库中内容同时设置缓存，以便提高下次请求响应速度。为了保证数据一致性，若涉及到写数据库相关操作，则会清空缓存。如图 5.1 所示。

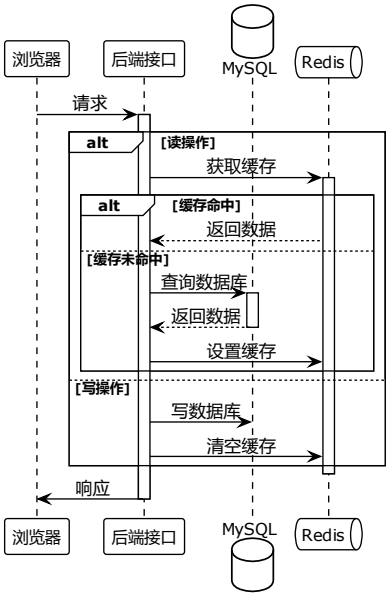

图 5.1 缓存工作过程

为了提升程序代码的复用能力，可以通过使用装饰器，将常用的功能封装成装饰器函数，然后在需要应用这些功能的函数上使用装饰器进行装饰。装饰器本质上是一种特殊的 Python 函数，它能够在不修改被装饰函数源代码的情况下，为其增加额外的功能。

```
def __call__(self, func: Callable) -> Callable:
    @wraps(func)
    def wrapper(re_self, request, *args: Any, **kwds: Any):
        # 函数调用前的逻辑
        ...
        # 函数调用
        res = func(re_self, request, *args, **kwds)
        # 函数调用后的逻辑
        ...
        return res
    return wrapper
```

这里的__call__方法使得装饰器实例能够像函数一样被调用。它接受一个函数作为参数，并返回一个新的函数 wrapper 作为装饰器的结果。

```
if self._cache_type == "w":
    res = func(re_self, request, *args, **kwds)
    self._redis.flushdb(1)
    return res
```

在函数调用前，如果 self._cache_type 等于"w"，表示涉及到写数据库的操作，那么先执行原始函数 func，然后清空缓存，最后返回函数的结果。

```
payload = f"{request.path}+{request.GET}"
cache_val = self._redis.get(payload)
```

如果没有涉及写数据库的操作，首先根据请求的路径和参数生成一个唯一的 payload 作为缓存键，然后使用 self._redis 对象从缓存中获取对应的值。

```
if not cache_val:
    response = func(re_self, request, *args, **kwds)
```

```
    if response.status_code == 200:
        if hasattr(response, "_headers"):
            headers = response._headers.copy()
        else:
            headers = {k: (k, v) for k, v in response.items()}
        response_triple = (response.data, response.status_code,
headers)
        self._redis.setex(payload, self._cache_timeout,
pickle.dumps(response_triple))
    return response
```

如果缓存未命中，即 cache_val 为空，那么调用原始函数 func 获取响应，并将响应的内容、状态码和头部信息存储为 response_triple。然后，将 response_triple 序列化为字符串，并以 payload 为键，存储在 Redis 缓存中，同时设置过期时间为 self._cache_timeout。最后，返回响应。

5.1.3 敏感词过滤功能实现

为了维护良好的网络环境，防止网络上出现违反国家法律法规的内容，本项目对评论内容增加了敏感词过滤功能。传统敏感词过滤常用方法有关键词匹配、正则表达式等，这些方法在处理少量敏感词时效率很高，一旦处理敏感词过多，性能会明显下降。本项目采用基于 DFA 的敏感词过滤算法。

DFA（Deterministic Finite Automaton）是一种有限状态自动机，用于识别正则语言。它是一种抽象的计算模型，可以描述由有限数量的状态、输入符号和转移函数组成的系统。对于敏感词过滤，我们可以将每个字符看作是 DFA 的一个状态，状态之间的转移表示字符的连接关系。通过遍历敏感词库，我们可以构建一个 DFA 的状态转移表。例如，有“我是张三”，“我是李四”，“坏人”三组敏感词，经过 DFA，可以得到如 5.2 的状态图。

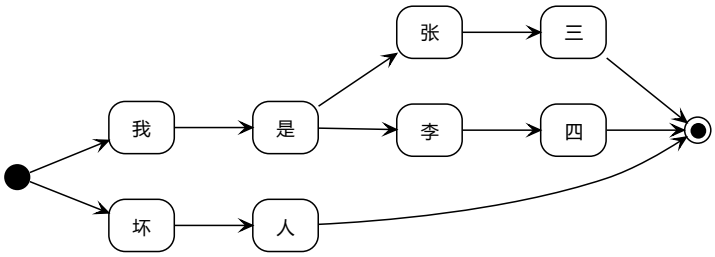

图 5.2 状态转移图

这样就将我们的敏感词库构建成了类似树的数据结构，如此以来，可以有效地减少敏感词的匹配范围，从而提升敏感词检测的效率。同时通过在叶子节点中设置表示位表示位来判断是否完成敏感词匹配。

例如，有“你好，我是张三，是一个坏人。”一句话，可以通过遍历每个字符来判断当前字符是否在状态转移表中，进而进行替换，实现敏感词过滤。

5.1.4 滑动拼图验证功能实现

滑动拼图验证是一种常见的人机验证机制，旨在识别用户是否为真实人类而不是自动化脚本或机器人。它的作用是防止恶意程序或机器人自动化执行某些操作，如批量注册、刷票、爬取网站内容等。本项目在登录注册时采用了滑动拼图验证。

首先，从背景图像中读取图像数据，然后获取图像的宽度和高度。接下来，生成拼图的位置坐标，限定在合理的范围内，防止取值越界。然后，根据坐标截取背景图像中的拼图部分，并将其转换为 base64 编码。同时，读取遮罩图像数据，将遮罩图像粘贴到背景图像上，再将合成后的图像转换为 base64 编码。最后，返回拼图图像的 base64 编码、合成后的图像的 base64 编码、拼图的 x 坐标和 y 坐标。如图 5.3 所示。

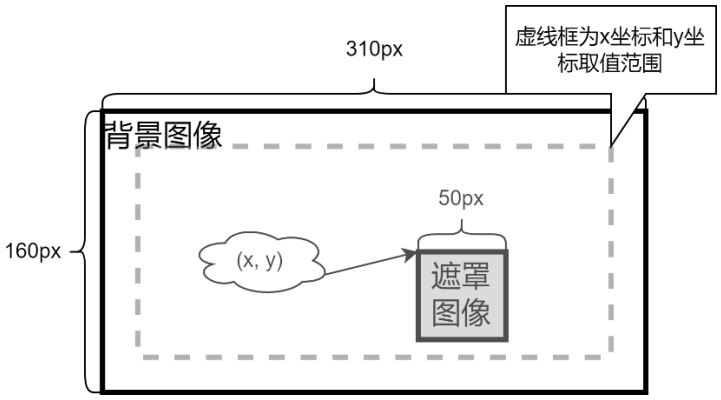

图 5.3 滑动拼图验证设计图

前端可以根据返回的数据构建验证页面。

5.2 系统实现

5.2.1 公共页面模块实现

公共页面包括公共主页，用户主页，图片详情和分区详情。

当用户打开网络相册是，首先进入的是公共主页，公共主页中展示用户上传的公开的并且通过审核的照片，同时左侧导航栏第二栏为分区列表，主页图片分页展示，一页展示为 8 个，如图 5.3 所示。

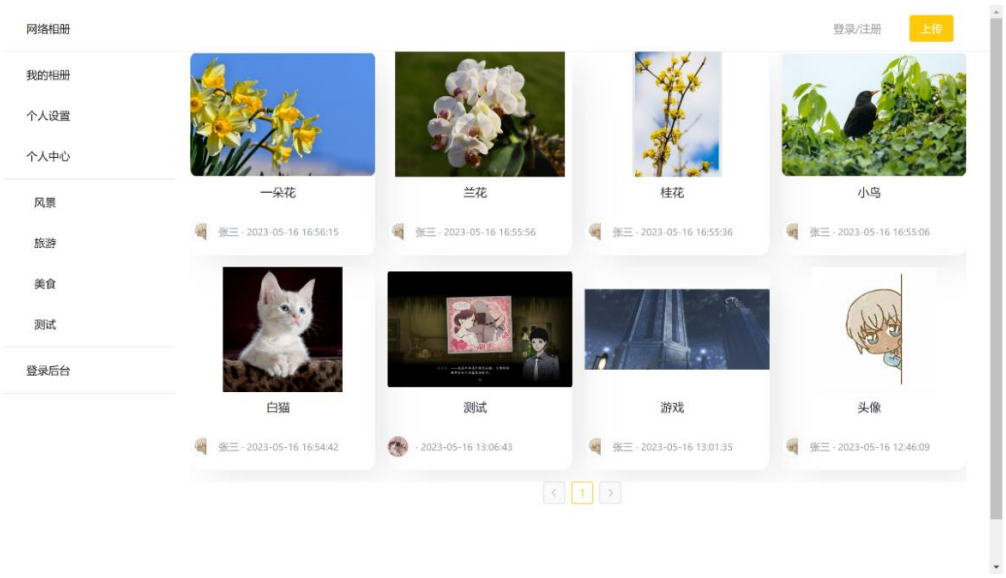

图 5.4 公共主页

当用户点击图片名称时，进入图片详情页，图片详情页中展示图片大图，上传者信息，评论列表以及部分图片信息，如点赞量，浏览量，简介，上传时间等，如图 5.4 所示。

图 5.5 图片详情

在公共主页点击用户头像后，进入用户主页，主页展示该用户上传的所有公开照片，点击这些照片可以进入图片详情页，以及用户的基本信息，如用户昵称，用户头像，用户性别等。点击公共主页左侧的分区列表进入分区详情，分区详情页展示不同分区照片，界面展示如图 5.5 所示。

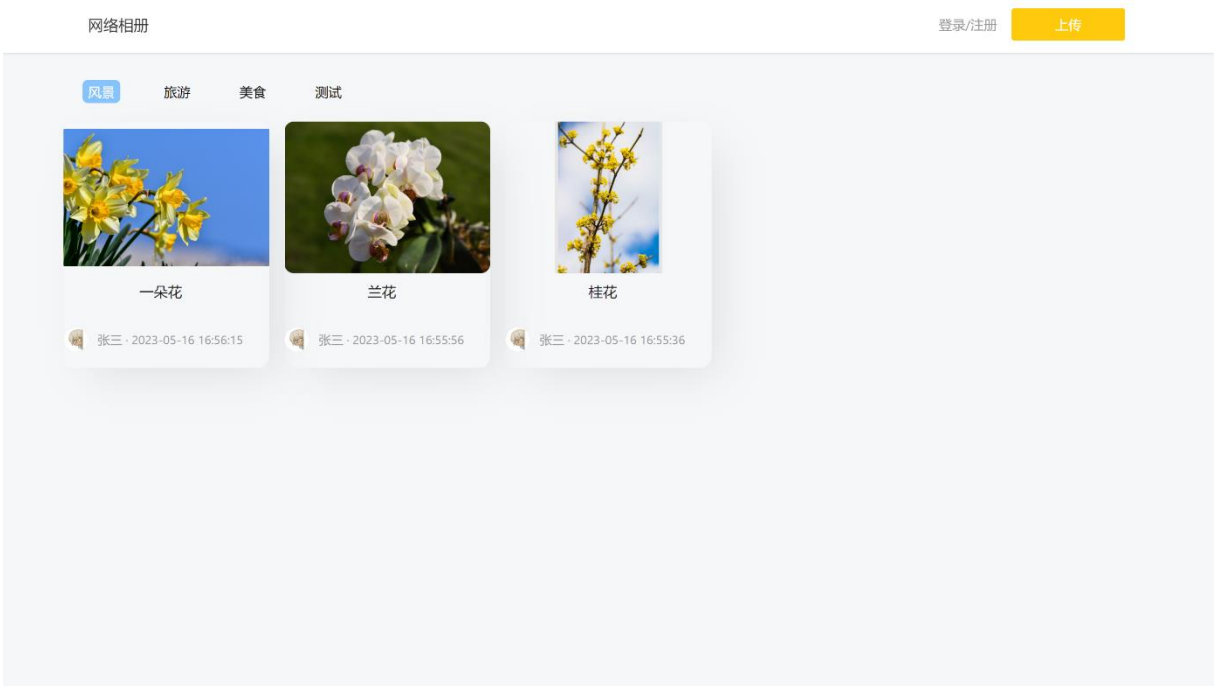

图 5.6 分区详情

公共页面模块部分前端代码如下：

```
<template>
    <div class="home" :style="initTheme()" v-title :data-
title="`${globalConfig.title}`">
        <home-header class="home-header"></home-header>
        <div class="home-content">
            <div class="home-left">
                <home-sidebar class="home-sidebar"></home-sidebar>
            </div>
            <div class="home-right">
                <div class="home-recommended">
                    <picture-item v-for="item in
pictureList" :info="item"></picture-item>
                </div>
                <n-pagination style="justify-content: center; " v-
model:page="page" :item-count="count" :page-size="8"
                    @update-page="pageChange" />
            </div>
        </div>
    </div>
</template>
......
```

特别的，当游客用户使用非游客用户功能时，则会弹出登录卡片，提示游客用户登录或者注册，有效提高网络相册的安全性。实现代码如下：

```
router.beforeEach((to, from, next) => {
    if (to.meta.auth && !storageData.get('access_token')) {
        if (from.name !== "Home") {
            router.push({ name: 'Login' });
```

```
            next();
        } else {
            loginStore.setLoginState(true);
        }
    } else if (to.meta.admin
&& !storageData.get('access_token_manage')) {
        router.push({ name: 'AdminLogin' });
        next();
    } else {
        next();
    }})
```

这段代码是一个路由守卫，在 Vue Router 中使用。它的作用是在每次路由跳转之前执行一些逻辑来控制路由的访问权限。

在代码的逻辑中，首先检查了目标路由对象是否具有 meta 字段，并且该字段中包含"auth"属性。这是为了检查是否需要身份验证才能访问目标路由。同时，还检查了存储中是否存在"access_token"，以确定用户是否已经登录。如果需要身份验证并且未登录，将用户重定向到登录页。

如果需要管理员权限并且管理员未登录，则会通过将管理员重定向到管理员登录页，并调用 next()继续执行路由跳转。

如果既不需要用户登录权限，也不需要管理员登录权限，则正常执行路由跳转。

5.2.2 登陆注册模块实现

用户可以通过该模块实现登陆系统或者注册成为网络相册用户。当用户在公共主页点击登录按钮时，会弹出登录注册页面，界面展示如图 5.6 所示。

图 5.7 登录注册页

用户输入用户名密码即可登入网络相册系统。登录功能部分后端代码如下：

```
user = User.objects.filter(username=username).first()
if not user:
    res.update(msg="用户不存在", code=2)
    return res.data
if user.is_freeze:
```

```
    res.update(msg="用户已被冻结", code=2)
    return res.data
if not PasswordUtil.verify(PasswordUtil(), password, user.password):
    res.update(msg="密码错误", code=2)
    return res.data
user.jwt_version += 1
payload = {"id": user.id, "jwt_version": user.jwt_version}
jwt_token = JwtTokenUtil().encode_user(payload)
user.save()
res.update(data={"username": user.username, "token": jwt_token})
```

首先判断用户是否存在或者已被冻结，若用户存在并且正常，则判断密码是否正确，此处用户的密码是以密文存储在数据库，提高系统安全性。用户登录成功后，会返回包含用户名和 token 的 json 数据给前端，前端将 token 存储在 localStorage 中，当浏览器向服务器发起请求时，会通过请求拦截器将 token 添加到请求头中。

当用户登录或注册时，需要先进行人机验证验证之后才能进行下一步，对于人机验证，本项目没有采用传统的验证码形式，而是采用滑动拼图验证。用户需要将滑块拖动到正确位置才能完成验证。滑动拼图验证的目的是为了增强网站或应用程序的安全性，防止恶意攻击和自动化机器人的登录尝试。传统的用户名和密码登录方式容易受到暴力破解、密码泄露等攻击，而滑块验证可以提供一种额外的验证层，增加登录的安全性。界面展示如图 5.7 所示。

图 5.8 滑动拼图验证

注册用户忘记密码可以在登录页点击找回密码，通过注册时填写的邮箱账号重置密码，后端会生成一个随机六位数验证码，并且将验证码发送给用户邮箱，同时保存在缓存中，用户凭借收到的验证码进行密码重置。界面展示如图 5.9 所示。

图 5.9 找回密码页面

5.2.3 用户信息模块实现

用户登录之后可以进入个人主页对个人信息进行修改，可选修改选项有用户头像，用户昵称，用户生日，地区等；对于用户头像修改，页面提供了实时预览功能，同时可以进行简单裁剪。在账号安全页面，可以对密码进行修改，密码修改过程与找回密码相同，需要用户输入注册邮箱。界面展示 5.10 所示。

图 5.10 修改用户信息页面

部分前端代码如下所示：

```
const modifyUserInfo = () => {
    const modifyForm: ModifyUserInfoType = {
        avatar: userInfo.value.avatar || '',
        nickname: userInfo.value.nickname || '',
        gender: userInfo.value.gender || '2',
        birthday: userInfo.value.birthday || '1970-01-01',
        regions: userInfo.value.regions || '',
    modifyUserInfoAPI(modifyForm).then((res) => {
        if (res.data.code === statusCode.success) {
```

```
            getUserInfo();
            notification.success({
                title: '修改成功',
                duration: 3000, );
            setTimeout(() => {
                router.go(0);
            }, 2000);}});
}
```

5.2.4 图片信息模块实现

用户打开个人主页后，可以管理已上传的图片包括编辑，删除图片等。当用户点击修改图片时，会弹出修改图片信息模态框，模态框中包含图片信息相关表单，此时表单中已经填充了图片的相关信息，用户对其进行修改保存即可。界面展示如图 5.11 所示。

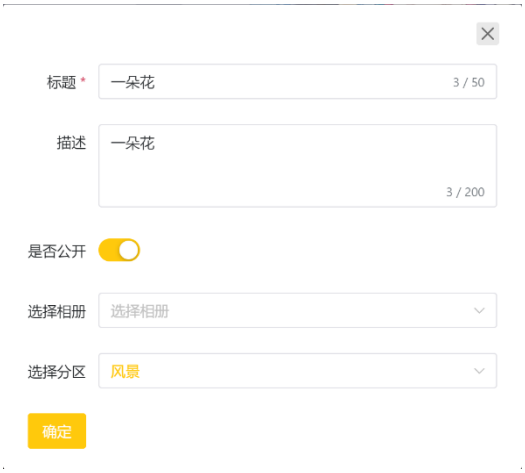

图 5.11 修改图片信息

在图片详情页面，登录用户可以对图片发起点赞。初始图片未点赞，点赞按钮未灰色，当用户发起点赞时，点赞按钮会变成亮色，同时点赞数量增加 1。由于点赞过程不需要刷新页面，因此点赞数量变化在前端完成，无需请求后端接口。部分代码如下所示：

```
const likeClick = () => {
    if (!is_like.value) {
        //调用点赞接口
        likeAPI(id).then((res) => {
            if (res.data.code === statusCode.success) {
                likeAnimation.value = 'like-active';
                like.value++;
            } )
    } else {
        cancelLikeAPI(id);
        like.value--;
    }
    is_like.value = !is_like.value;
}
```

当用户点击点赞按钮时，首先通过检查 is_like.value 的值来确定当前是否已经点赞。如果未点赞，则调用点赞接口，并传递参数 id。将 like.value 增加 1，表示点赞数加 1。已经点赞，则调用取消点赞接口，并传递参数 id。将 like.value 减少 1，表示点赞数减 1。最后，将 is_like.value 的值取反，表示点赞状态的切换。

登录用户可以在用户详情页发表评论和对评论进行回复，如果评论或者回复中包含敏感词，则会被替换成“*”。同时，若评论或者回复的内容是本人发表的，可以对评论进行删除；非本人发表的评论或回复则不会显示删除按钮。界面展示如图 5.12 所示。

图 5.12 评论回复详情

5.2.5 相册信息模块实现

登录用户可以在个人主页对相册进行增加、修改和删除操作，界面展示如图 5.13 所示。

图 5.13 相册信息页面

用户单机创建相册按钮后，会弹出创建相册模态框，此时用户可以上传相册封面，相册名称，相册描述。创建相册页面如图 5.14 所示。

图 5.14 创建相册页面

创建相册部分代码如下所示：

```
<n-modal style="width: auto;" v-model:show="createActive"
preset="card" title="创建相册">
    <cover-uploader :cover="createAlbumInfo.cover"
@finish="finishUpload"></cover-uploader>
    <n-form>
        <n-form-item label="相册名称" required>
            <n-input v-model:value="createAlbumInfo.name"
placeholder="请输入相册名称"></n-input>
        </n-form-item>
        <n-form-item label="相册描述" required>
            <n-input v-model:value="createAlbumInfo.desc"
placeholder="请输入相册描述"></n-input>
        </n-form-item>
        <n-button type="primary" @click="createAlbum">保存</n-button>
    </n-form>
</n-modal>
```

用户点击相册时，会进入相册详情页面，相册详情页左侧展示相册中照片，对图片展示进行了分页处理，一页展示 8 个，用户可以对相册中的图片进行删除操作；相册详情页右侧展示相册相关信息，如相册封面，相册名称，相册描述等。界面展示如图 5.15 所示。

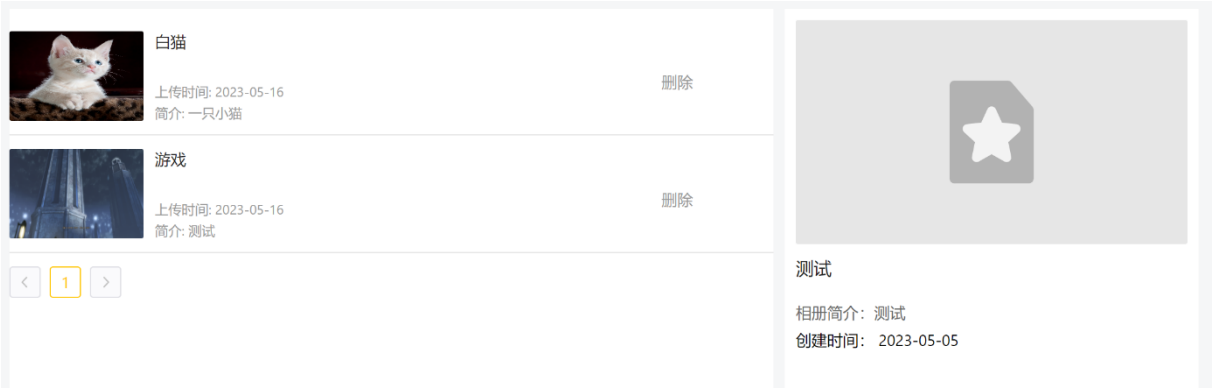

图 5.15 相册展示页

5.2.6 上传管理模块实现

对于本项目，上传图片功能为核心功能，用户登录后可以打开上传页面，上传页面中用户可以上传图片，填写图片相关信息。界面展示如图 5.16 所示。

图 5.16 图片上传页面

前端规定了图片上传允许的大小以及图片格式，对于不符合的格式不允许上传，具体实现代码如下：

```
const beforeUploadCover = async (options: any) => {
    const file = options.file;
    const isJpgOrPng =
        file.type === "image/jpeg" || file.type === "image/png";
    if (!isJpgOrPng) {
        notification.error({
            title: '上传失败',
```

```
            content: "文件只支持 jpg/jpeg/png 格式",
            duration: 5000,
        }); }
    const isLtMaxSize = file.file.size / 1024 / 1024 <
globalConfig.maxImgSize;
    if (!isLtMaxSize) {
        notification.error({
            title: '上传失败',
            content: `图片大小不能超过${globalConfig.maxImgSize}M`,
            duration: 5000,
        }); }
    return isJpgOrPng && isLtMaxSize;
}
```

这段代码是一个用于上传图片前的验证函数，使用了异步函数的形式。函数 beforeUploadCover 接收一个参数 options，这个参数包含了上传文件的相关信息。首先，从 options 中获取上传的文件对象，将其赋值给变量 file。接下来，通过判断文件的类型，检查是否为 JPEG 或 PNG 格式的图片。通过比较文件大小是否小于 globalConfig.maxImgSize 来判断图片大小是否超过了限制。最后，函数返回一个布尔值，如果文件类型和大小都符合要求，则返回 true；否则，返回 false。

当用户上传文件后，后端会将文件保存至 IPFS 分布式文件存储系统上，实现代码如下：

```
class UploadPhotoToIPFS(views.APIView):
    parser_classes = [MultiPartParser]
    def post(self, request):
        res = JsonResponse()
        images = request.FILES.items()
        key, image = next(images)
        ipfs_client = ipfs.connect("/ip4/127.0.0.1/tcp/5001")
        image = ipfs_client.add(image)
        image_hash = image["Hash"]
        res.update(data={"url": image_hash})
        return res.data
```

后端首先获取前端上传的文件信息，储存在内存中；之后建立与 IPFS 节点的连接。使用 ipfs.connect()方法创建一个 IPFS 客户端实例，连接到本地节点的 5001 端口。将图片上传到 IPFS 网络，此时会返回一个包含上传结果的对象，之后将对象中的 Hash 值返回给前端，用作表单内容。

特别的，用户上传的公开图片初始状态为待审核状态，需要经过管理员审核才能变成正常状态。

5.2.7 系统管理模块实现

系统管理模块是独立于网络相册的功能模块，用于管理整个网络相册的所有内容。采用独立的管理员账号登录，管理员账户不支持注册，写死在数据库中。系统管理主要包括用户管理，图片管理，评论管理，分区管理，敏感词管理等。界面展示如图 5.17 所

示。

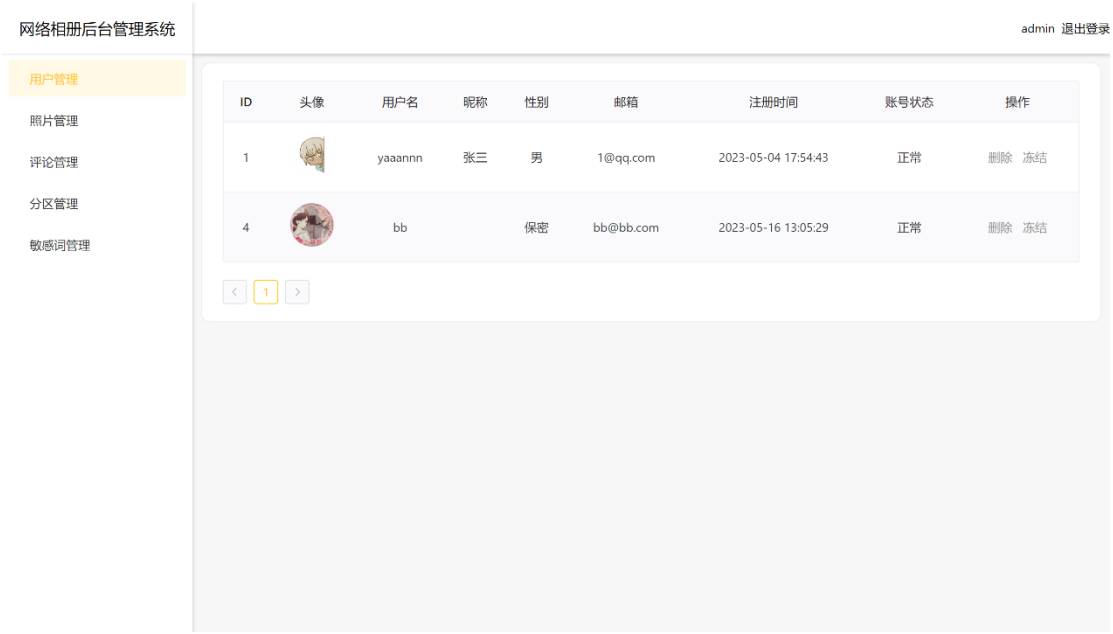

图 5.17 系统管理页面

在用户管理中，管理员可以查看所有用户的基本信息，可以删除或者冻结用户；在照片管理中，管理员主要负责审核照片，用户初上传的照片为待审核状态，通过系统管理员审核后才会变为正常状态；分区管理中，管理员可以添加，修改和删除分区，分区管理界面如图 5.18 所示。

图 5.18 分区管理页面

6 系统测试

6.1 单元测试

单元测试是软件测试中的一种测试级别，旨在验证软件中最小的可测试单元功能是否正常。单元通常是指一个函数、方法、类或模块，它是软件的最小独立部分。

本项目功能模块众多，列出所有功能单元的测试结果篇幅众多，因此对部分重要功能模块进行测试。

1. 敏感词过滤测试

这里借助 python 单元测试工具 unittest 对其进行测试。首先编写测试代码如下：

```
class TestDFAFilter(unittest.TestCase):
    def setUp(self):
        self.filter = DFAFilter()
    def test_filter_message(self):
        filtered_message = self.filter.filter("这是一段包含敏感词的文本，敏感词应该被过滤。")
        self.assertEqual(filtered_message, "这是一段包含***的文本，***应该被过滤。")
    def test_filter_message_no_keywords(self):
        filtered_message = self.filter.filter("这是一段正常的文本。")
        self.assertEqual(filtered_message, "这是一段正常的文本。")
    def test_filter_message_with_replacement(self):
        filtered_message = self.filter.filter("这是一段包含敏感词的文本，敏感词应该被过滤。", repl="#")
        self.assertEqual(filtered_message, "这是一段包含###的文本，###应该被过滤。")
```

上述代码中 setUp 方法在每个测试方法之前执行，用于创建一个 DFAFilter 实例，以便在每个测试方法中使用。剩下的四个 test 开头方法分别是测试用例，其中使用 assertEqual 断言方法判断测试结果与预期结果是否相同。代码运行结果如下：

```
./test.py::TestDFAFilter::test_filter_message Passed
./test.py::TestDFAFilter::test_filter_message_no_keywords Passed
./test.py::TestDFAFilter::test_filter_message_with_replacement
Passed

Total number of tests expected to run: 3
Total number of tests run: 3
Total number of tests passed: 3
Total number of tests failed: 0
Total number of tests failed with errors: 0
Total number of tests skipped: 0
```

敏感词过滤功能正常，通过单元测试。

2. 管理员登录测试

编写测试用例代码如下：

```
class AdminAPITestCase(APITestCase):
    def setUp(self):
        self.admin = Admin.objects.create(
            username="admin",
            password="admin",
            email="1@qq.com",
        )

    def test_admin_login(self):
        url = reverse("admin_login")
        data = {
            "username": "admin",
            "password": "admin",
        }
        response = self.client.post(url, data)
        self.assertEqual(response.status_code, status.HTTP_200_OK)
        self.assertIn("token", response.data["data"])
```

该代码是一个使用 Django 的 APITestCase 编写的测试类 AdminAPITestCase，用于测试管理员登录的 API。setUp 方法在每个测试方法之前执行，用于创建一个管理员实例，以便在每个测试方法中使用。

test_admin_login 方法是一个测试用例，用于测试管理员登录的 API。它使用 reverse 方法获取登录 API 的 URL，构造一个包含用户名和密码的数据字典，然后使用 self.client.post 方法向 API 发送 POST 请求。最后，使用 self.assertEqual 断言方法验证响应的状态码是否为 200，并使用 self.assertIn 断言方法验证响应的数据中是否包含 token 字段。测试结果如下：

```
System check identified no issues (0 silenced).
.
----------------------------------------------------------------------
Ran 1 test in 0.033s

OK
```

测试结果符合预期，管理员登录功能正常。

6.2 性能测试

性能测试的主要目的是评估系统、应用程序或组件在不同负载条件下的性能和稳定性。性能测试常用于确定系统的性能指标，如响应时间、吞吐量、并发性能等。通过性能测试，可以了解系统在正常负载和峰值负载下的表现，从而评估系统是否满足性能需求和用户期望。

通过模拟高负载条件和异常情况，性能测试可以验证系统的稳定性和容错能力。压力测试可以帮助评估系统在负载峰值、异常请求和异常情况下的表现，以确定系统是否能够正常工作并保持稳定。

本项目使用了接口缓存技术，将经常访问的 api 请求响应存放在数据库中，大大提高了请求响应速度。为了详细对比接口缓存技术对请求响应的影响，编写如下代码对常

用接口进行单点请求测试：

```
def run_performance_test(url, num_requests):
    total_time = 0
    min_time = float('inf')
    max_time = 0
    for _ in range(num_requests):
        start_time = time.perf_counter()
        response = requests.get(url)
        end_time = time.perf_counter()
        request_time = end_time - start_time
        total_time += request_time
        if request_time < min_time:
            min_time = request_time
        if request_time > max_time:
            max_time = request_time
    avg_time = total_time / num_requests
```

上述代码利用 perf_counter 方法对 python 代码执行时间进行记录，间接计算访问时间，两种情况运行结果如图 6.1 所示。

```
请求URL: http://127.0.0.1:8000/api/v1/photo/public
总请求数: 100
总响应时间: 2335.704 ms
平均响应时间: 23.357 ms
最小响应时间: 20.151 ms
最大响应时间: 31.835 ms
```

(a) 未使用接口请求

```
请求URL: http://127.0.0.1:8000/api/v1/photo/public
总请求数: 100
总响应时间: 715.132 ms
平均响应时间: 7.151 ms
最小响应时间: 5.751 ms
最大响应时间: 11.373 ms
```

(b) 使用接口请求

图 6.1 请求响应时间对比

由上图平均响应时间对比可知，使用接口缓存技术后，平均响应时间变为原来 1/3，有效提升接口性能，提升用户体验。

下面对该接口进行压力测试，使用 python 开源工具 locust 对其进行测试，编写如下测试脚本：

```
class Test(TaskSet):
    @task(1)
    def test_queryMessage(self):
        post_url = '/api/v1/photo/public'
        self.client.get(post_url)

class WebsiteUser(HttpUser):
    host = "http://127.0.0.1:8000"
    tasks = [Test]
```

执行测试，设置最大访问用户数为 50，每秒增加 1 用户，进行五分钟左右压力测试。响应时间如图 6.2 所示。

Method	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	RPS
GET	/api/v1/photo/public	29582	0	515	6	989	89.7

图 6.2 响应时间

由图可知，对于本地服务器，模拟 50 名用户多线程访问，服务器每秒处理请求次数（RPS）约为 90，近三万次请求平均响应时间为 515ms，压力测试表现良好，符合要求。

[redacted]

总结

本篇论文旨在设计和实现一个前后端分离的网络相册系统，提供用户方便地上传、管理和共享照片的功能。通过该系统，用户可以创建相册、上传照片，并设置照片的访问权限。系统提供了用户管理、相册管理、照片管理等核心功能，系统将用户照片存储在分布式网络中，符合 WEB3.0 去中心化的概念。

在论文中，首先介绍了网络相册选题的背景和意义，和国内外研究现状，解释了为什么需要设计和实现这样一个系统。网络相册可以让用户方便地存储和管理大量的照片，并通过网络与他人分享。随着手机和数码相机的普及，人们拍摄的照片数量越来越多，传统的存储方式已经无法满足需求，因此网络相册成为一种重要的解决方案。

接着，论文详细介绍了网络相册系统的需求分析和设计。通过对用户需求的调研和分析，确定了系统的主要功能和特性。系统需要提供用户注册、登录、相册创建、照片上传、照片评论等功能。在设计阶段，采用了分层架构和模块化设计的思路，将系统拆分为多个模块，便于开发和维护。同时，使用了流行的 Web 开发框架和技术，如 Django、VUE 等，以实现系统的前后端交互和功能实现。

随后，论文描述了系统的具体实现过程。详细介绍了系统各个模块的实现细节，包括公共页面模块、登陆注册模块、用户信息模块等。

从开始选题，到需求分析，再到系统实现，经历了一次完整的软件开发流程，受益匪浅。由于时间、技术能力、实践能力的限制，本网络相册任有很多不足之处，具体有以下几点：

1. 对于系统部分代码仍有冗余，不利于后期修改与维护，仍然需要进一步优化。
2. 针对接口请求功能的实现，可以在操作 redis 数据库添加分布式锁，避免缓存击穿，缓存雪崩等危险出现；同时，涉及到读数据库操作时可以按需清理缓存而不是清空缓存。
3. 图片详情页面再看人数功能暂时没有实现，原计划打算利用 websockets 长连接技术实现，但本人能力有限，仍需继续学习。
4. 后台管理功能尚不完善，本项目的后台管理比较为简单，只支持简单操作，后续应该继续完善功能。
5. 系统功能可以进一步扩展。未来可以考虑加入更多的高级功能，如图片编辑、标签管理等，以提升系统的功能和竞争力。

[redacted]

参考文献

[1] 毛家麒.基于 JSP 的网络相册系统实现[J].信息记录材料,2019,20(06):118-119.DOI:10.16009/j.cnki.cn13-1295/tq.2019.06.068.

[2] 杨磊.基于 J2EE 网络相册管理系统的设计与实现[D].电子科技大学,2014.

[3] 宋之瞻.基于 SSH 框架的网络电子相册系统设计与实现[D].江西财经大学,2016.

[4] 徐雅琴,何成万,严柯.基于 REST 服务的电子相册的实现[J].武汉工程大学学报,2017,39(01):78-82.

[5] 胡阳. Django 企业开发实战[M].北京:人民邮电出版社,2019:97-99.

[6] 管权,焦祥璞.基于 SSH 的网络相册系统的设计[J].科技广场,2013,No.139(06):245-248.

[7] 王萌.基于云的 HTML5+JQ 跨平台交互式电子相册研究[D].西安理工大学,2018.

[8] 陈林.基于 Django 的英语在线视频教学系统的设计与实现[D]. 厦门大学, 2014.

[9] 郑苑丹.基于 OpenCV 的电子相册制作 APP 的设计与实现[J].信息与电脑(理论版),2021,33(02):75-77.

[10] 马德锦.REST 基于 MVRC 模式下的在线网络相册研究[D]. 南京邮电大学, 2013.

[11] 崔娟, 章恒, 马尧,等. 基于 Spring Security 框架的前后端分离软件平台构建的研究[J]. 科学技术创新, 2022(000-004).

[12] Antonio Melé. Django 4 By Example[M]. Birmingham:Packt,2022:616-639.

[13] Mestre P , Rui M , Melo-Pinto P , et al. securing restful web services using multiple json web tokens[J]. 2018.

[14] Vamsi K M , Lokesh P , Reddy K N , et al. Visualization of Real World Enterprise Data using Python Django Framework[J]. IOP Conference Series Materials Science and Engineering, 2021, 1042(1):012019.

[15] Alnavar K , Kumar R U , Babu C N . Document Parsing Tool for Language Translation and Web Crawling using Django REST Framework[J]. Journal of Physics: Conference Series, 2021, 1962(1):012018 (10pp).

致谢

[redacted]