

浙江大学

本科实验报告

课程名称 B/S 体系软件设计

实验项目 智能图片管理网站

姓 名 蒋翼泽

学 号 3230102996

电子邮箱 3230102996@zju.edu.cn

联系电话 13456671250

指导老师 胡晓军

实验日期 2025 年 11 月 17 日

目录

1 项目背景	3
2 系统需求分析	3
2.1 功能性需求分析	3
2.2 非功能性需求分析	4
3 系统技术选型与架构设计	4
3.1 系统架构	4
3.2 技术栈选型	5
3.3 前端主要技术介绍	6
3.4 后端主要技术介绍	6
4 数据库设计分析	6
4.1 数据库简介	6
4.2 数据库表描述	7
4.2.1 1. Users (用户表)	7
4.2.2 2. Images (图片表)	8
4.2.3 3. Albums (相册表)	8
4.2.4 4. Tags (标签表)	9
5 功能实现分析	9
5.1 用户注册 (API: POST/api/auth/register)	9
5.2 用户登录 (API: POST/api/auth/login)	10
5.3 图片上传 (API: POST/api/images/upload)	10
5.4 图片搜索 (API: GET/api/images/search)	10
5.5 mcp 文本检索 (API: GET/api/ai/search)	11
6 系统界面原型	11
6.1 认证界面 (登录/注册)	11
6.2 图库主页 (GalleryPage)	11
6.3 图片详情 (轮播模态框)	11

7 增强功能实现	12
8 部署方案	12

1 项目背景

本项目是《B/S 体系软件设计》课程的期中项目，旨在设计和实现一个功能完备的“智能图片管理网站”。

根据需求文档，本项目需要支持多用户，每个用户登录后可以上传、管理、搜索和编辑自己的图片。系统需要能自动提取图片的辅助信息（如 EXIF）、生成缩略图，并允许用户添加自定义标签。此外，系统还需要提供 AI 分析和 `mcp` 文本检索等高级功能。

同时，网站界面需要对用户友好，样式需适配手机移动端，能够在手机浏览器和微信等应用内置的浏览器中友好显示。项目最终需要使用 Docker Compose 进行容器化部署。

本文档是该项目的系统设计报告，详细阐述了系统的需求分析、总体架构、数据库设计、功能实现和界面原型。

2 系统需求分析

2.1 功能性需求分析

该项目是一个 B/S 架构的 Web 应用，主要实现以下功能性需求：

1. 用户管理：

- 用户注册：需要提供用户名、Email 和密码（6 位以上）。
- 信息校验：确保用户名和 Email 在系统中唯一。
- 用户登录：通过用户名和密码进行身份验证。

2. 图片管理：

- 图片上传：支持 PC 端浏览器拖拽或点击上传图片（可多选）。
- 信息提取：自动读取并保存图片的 EXIF 信息（时间、地点、分辨率等）。
- 分类管理：允许用户添加自定义分类标签，方便检索。
- 相册功能：允许用户创建相册，并将图片归类到不同相册。
- 图片删除：提供删除功能。

3. 图片浏览与搜索：

- 缩略图显示：高效加载和显示缩略图
- 条件查询：提供界面，能根据各种条件（如标签、日期、文件名）查找图片。
- 轮播展示：支持选择一组图片进行轮播/幻灯片显示。

4. 图片编辑：

- 提供简单的在线编辑功能，如裁剪、旋转、修改色调等。

5. 增强功能:

- AI 智能分析: 调用 AI 模型分析图片, 提供多类型的标签 (如风景、人物等)。
- **mcp** 文本检索: 调用大模型对图片进行“文本-图像”检索。

2.2 非功能性需求分析

1. 用户体验 (UX):

- 界面简洁友好, 交互逻辑清晰。
- 系统应保证运行稳定, 响应流畅。

2. 跨平台兼容性:

- 网站界面需要采用响应式设计, 适配 PC 和手机移动端。
- 确保在主流浏览器 (Chrome, Firefox) 及微信内置浏览器中友好显示。

3. 数据管理:

- 图片元数据 (用户信息、标签、EXIF 等) 存储在 MySQL 数据库中。
- 图片文件本体 (原图、缩略图) 应与数据库分离, 采用对象存储。
- 保证数据库和文件存储的安全性、一致性。

4. 部署与维护:

- 项目代码使用 Git 进行版本管理。
- 项目需提供 **docker-compose.yml** 文件, 实现一键容器化部署。

3 系统技术选型与架构设计

本项目采用前后端分离的现代 Web 开发技术, 后端编写 RESTful API 供前端调用。

3.1 系统架构

- **前端 (Client)**: 一个单页面应用 (SPA), 负责所有用户界面的渲染和交互逻辑。
- **后端 (Server)**: 一个提供 RESTful API 的 Go 服务, 负责业务逻辑、数据处理、用户认证和文件存储。
- **文件存储**: 使用独立的对象存储服务 (MinIO), 实现图片文件的可靠存储和高效分发。
- **数据库**: 采用 MySQL 数据库存储用户信息、图片元数据、相册、标签等结构化数据。
- **AI 服务**: 一个独立的 Python 微服务, 用于处理 AI 分析和 **mcp** 检索任务, 与主后端异步通信。

3.2 技术栈选型

类别	技术	备注
前端	React.js	业界领先的 UI 库，组件化开发。
	Zustand	轻量级的全局状态管理库。
	TailwindCSS	Utility-First 的 CSS 框架，快速构建响应式布局。
后端	Go (Golang)	高性能的静态编译语言。
	Gin	Go 语言中流行的高性能 Web 框架。
	GORM	Go 语言的全功能 ORM 库。
	golang-jwt/jwt	用于处理 JWT (JSON Web Tokens) 认证。
数据库	MySQL 8.0	遵照需求的 SQL 数据库。
文件存储	MinIO	S3 兼容的开源对象存储。
AI 服务	Python (FastAPI)	用于 AI 模型分析和 mcp 检索。
部署	Docker Compose	遵照需求的容器化部署方案。
版本控制	Git	遵照需求的版本控制系统。

表 1: 技术栈选型表

3.3 前端主要技术介绍

本项目前端主要使用 **React**。React 是一套用于构建用户界面的渐进式 JavaScript 库，其核心是组件化。通过组件，我们可以将 UI 拆分为独立可复用的部分，使开发和维护变得高效。配合使用 **TailwindCSS**，我们可以快速构建美观且完全响应式的界面，满足 PC 和移动端的适配需求。**Zustand** 作为一个轻量级状态管理器，用于处理跨组件的全局状态（如当前用户信息、图片列表等）。

3.4 后端主要技术介绍

本项目后端主要采用 **Go (Golang)** 语言，并选用 **Gin** 作为 Web 框架。Go 是由 Google 开发的一种静态强类型、编译型语言，它天生支持高并发 (Goroutines)，非常适合构建高吞吐量的网络服务。

Gin 是一个用 Go 编写的高性能 Web 框架，它提供了路由、中间件、JSON 绑定等核心功能，API 简洁且性能卓越。对于用户认证，我们将采用 **JWT (JSON Web Tokens)** 方案，并使用 [golang-jwt/jwt](#) 库来生成和验证 Token，实现无状态的身份验证。在数据库交互方面，使用 **GORM** 作为 ORM，它提供了强大的数据库操作封装和自动迁移功能，能极大提升开发效率。

4 数据库设计分析

4.1 数据库简介

根据需求分析，我们暂定设计五张核心表：**Users** (用户表), **Images** (图片表), **Albums** (相册表), **Tags** (标签表)，以及两张用于多对多关系的 **Album_Images** 和 **Image_Tags** 联结表。

4.2 数据库表描述

4.2.1 1. Users (用户表)

字段名	类型	描述	备注
user_id	INT	用户的唯一 ID	主键, 自增
username	VARCHAR(255)	用户的唯一用户名	非空, 唯一
email	VARCHAR(255)	用户的唯一 Email	非空, 唯一
password_hash	VARCHAR(255)	哈希后的密码	非空, 长度 >6
created_at	DATETIME	注册时间	默认当前时间

4.2.2 2. Images (图片表)

字段名	类型	描述	备注
image_id	INT	图片的唯一 ID	主键, 自增
user_id	INT	所属用户的 ID	外键, 关联 Users(user_id)
file_name	VARCHAR(255)	原始文件名	
storage_path	VARCHAR(255)	在对象存储中的路 径/Key	非空
thumbnail_path	VARCHAR(255)	缩略图的存储路 径/Key	
resolution	VARCHAR(50)	图片分辨率	如”1920x1080”
location	VARCHAR(255)	地理位置	(从 EXIF 提取)
exif_data	JSON	原始 EXIF 数据	
uploaded_at	DATETIME	上传时间	默认当前时间

4.2.3 3. Albums (相册表)

字段名	类型	描述	备注
album_id	INT	相册的唯一 ID	主键, 自增
user_id	INT	所属用户的 ID	外键, 关联 Users(user_id)
album_name	VARCHAR(255)	相册名称	非空
description	TEXT	相册描述	

4.2.4 4. Tags (标签表)

字段名	类型	描述	备注
tag_id	INT	标签的唯一 ID	主键, 自增
tag_name	VARCHAR(255)	标签名称	非空, 唯一

(注: *Album_Images* 和 *Image_Tags* 为联结表, 用于处理多对多关系)

5 功能实现分析

这里给出核心 API 接口的实现思路。所有 API 均以 `/api` 为前缀, 并受 JWT 认证保护 (登录/注册除外)。

5.1 用户注册 (API: `POST/api/auth/register`)

主要参数 ‘Object’ 注册界面提交的信息, 包括 `username`, `email`, `password`。

返回值类型 ‘Object’ 成功则返回用户信息和 `access_token`; 失败则返回错误信息 (如“Email 已存在”)。

简介 用于创建新用户。后端需对 `username` 和 `email` 进行唯一性校验, 并对 `password` 进行哈希加密处理 (如 `bcrypt`)。

5.2 用户登录 (API: POST/api/auth/login)

主要参数	‘Object’	登录界面提交的 <code>username</code> 和 <code>password</code> 。
返回值类型	‘Object’	成功则返回用户信息和 <code>access_token</code> ；失败则返回“用户名或密码错误”。
简介	用于用户的登录认证。后端校验密码，成功后签发 JWT。	

5.3 图片上传 (API: POST/api/images/upload)

主要参数	‘FormData’	包含 <code>file</code> (图片文件) 以及可选的元数据 <code>tags:string[]</code> , <code>albumId:int</code> 。
返回值类型	‘Object’	返回新创建的 ‘Image’ 对象的 JSON 数据。
简介	关键流程。后端接收文件，(1) (Go routine) 提取 EXIF、生成缩略图，(2) 上传原图和缩略图到 MinIO，(3) 将元数据和存储路径写入 MySQL 数据库。	

5.4 图片搜索 (API: GET/api/images/search)

主要参数	‘Query Params’	如 <code>?q=...</code> (模糊搜索) 或 <code>?tag=...</code> (按标签) 或 <code>?date=...</code> (按日期)。
返回值类型	‘Array<Image>’	返回符合条件的图片对象数组，支持分页。
简介	用于对图片进行多条件查询。	

5.5 mcp 文本检索 (API: GET/api/ai/search)

主要参数 ‘Query Param’
?prompt=... (用户输入的描述性文本, 如“海边的日落”。

返回值类型 ‘Array<Image>’
返回 AI 认为最相关的图片对象数组。

简介 (增强功能) 主后端将请求转发给 AI 服务, 使用 CLIP 等模型将文本转换为向量, 在向量数据库中进行相似性搜索。

6 系统界面原型

系统界面原型设计如下所示, 将采用简洁、现代的风格。

6.1 认证界面 (登录/注册)

- 采用居中卡片式设计。
- 提供“登录”和“注册”两个表单的切换。
- 包含必要的输入框(用户名、Email、密码)和前端实时校验。

6.2 图库主页 (GalleryPage)

- 登录后的主界面。
- 左侧 (FilterSidebar): 提供按相册、标签、日期的筛选器。
- 顶部 (Header): 提供全局搜索框(支持文本和 mcp)、上传按钮。
- 中部 (ImageGrid): 采用响应式网格布局, 无限滚动加载图片缩略图。

6.3 图片详情 (轮播模态框)

- 点击图库中的缩略图后, 以模态框(Modal)形式弹出。
- 居中显示大图, 提供左右箭头用于在当前列表中轮播切换。
- 侧边显示该图片的详细信息(EXIF、标签、所属相册)以及“编辑”、“删除”按钮。

7 增强功能实现

如 2.1 和 5.5 所述，AI 分析和 mcp 检索是本项目的增强功能。

- **AI 智能分析 (增强 1)**: 在图片上传成功后 (见 5.3)，主后端服务将 `image_id` 放入一个异步任务队列 (如 RabbitMQ 或 Redis Queue)。AI 服务 (Python/FastAPI) 监听此队列，获取任务后从 MinIO 下载图片，使用预训练的 ResNet 或 ViT 模型进行分类，并将生成的 AI 标签 (如“天空”，“建筑”，“狗”) 写回数据库 `Image_Tags` 表。
- **mcp 文本检索 (增强 2)**: AI 服务将使用 CLIP 模型对所有上传的图片提取图像向量 (Embedding)，并存入一个向量数据库 (如 Qdrant 或 Milvus)。当用户发起文本搜索时 (见 5.5)，AI 服务将文本同样转换为向量，并在向量数据库中执行高效的相似性查询，返回最匹配的图片。

8 部署方案

我们将使用 `docker-compose.yml` 来编排和管理所有服务，确保一键启动和隔离的环境。

```
1 # docker-compose.yml (概要)
2 version: '3.8'
3
4 services:
5     # 1. 前端 (使用 Nginx 托管 React 静态文件)
6     frontend:
7         build: ./frontend
8         ports:
9             - "80:80" # http 端口
10        depends_on:
11            - backend
12
13     # 2. 后端 (Go/Gin API)
14     backend:
15         build: ./backend # Go 应用的 Dockerfile
16         ports:
17             - "8080:8080" # Go 应用通常使用 8080 端口
18         environment:
19             - DATABASE_URL=mysql://user:password@tcp(db:3306)/dbname
20             - MINIO_ENDPOINT=minio:9000
21             - MINIO_ACCESS_KEY=...
22             - MINIO_SECRET_KEY=...
23             - JWT_SECRET=your-jwt-secret
24         depends_on:
```

```

25      - db
26      - minio
27
28 # 3. 数据库 (MySQL)
29 db:
30   image: mysql:8.0
31   restart: always
32   environment:
33     - MYSQL_ROOT_PASSWORD=your-root-password
34     - MYSQL_DATABASE=dbname
35   volumes:
36     - mysql-data:/var/lib/mysql # 数据持久化
37   ports:
38     - "3306:3306"
39
40 # 4. 对象存储 (MinIO)
41 minio:
42   image: minio/minio
43   restart: always
44   ports:
45     - "9000:9000" # S3 API
46     - "9001:9001" # Web Console
47   volumes:
48     - minio-data:/data
49   environment:
50     - MINIO_ROOT_USER=...
51     - MINIO_ROOT_PASSWORD=...
52   command: server /data --console-address ":9001"
53
54 # 5. (可选) AI 服务 (Python/FastAPI)
55 ai_service:
56   build: ./ai_service
57   ports:
58     - "8000:8000"
59
60 volumes:
61   mysql-data:
62   minio-data:

```

Listing 1: `docker-compose.yml` (概要)