《图片管理网站》

——系统设计报告

1. 引言

1.1. 编写目的

本文档的目的是详细阐述“图片管理网站”系统的设计方案。它定义了系统的总体架构、功能需求、模块划分、接口设计和数据库结构。

1.2. 文档范围

本文档范围涵盖“图片管理网站”的所有核心功能：

- 用户认证系统（注册、登录）
- 图片管理（上传、删除、查询）
- 元数据处理（EXIF 提取、缩略图生成）
- 标签系统（自定义标签、自动标签）
- 搜索系统（多条件检索）
- 增强功能（AI 分析、MCP 接口）的设计方案

1.3. 读者对象

- **项目经理**：用于了解项目范围、进度和功能划分。
- **软件开发人员**：作为详细设计和编码实现的依据。
- **软件测试人员**：作为编写测试用例和进行系统测试的依据。
- **系统维护人员**：用于了解系统架构和部署，方便后续维护。

1.4. 术语与缩写解释

缩写、术语	解释
API	Application Programming Interface (应用编程接口)
JWT	JSON Web Token (JSON Web 令牌)，用于身份认证
EXIF	Exchangeable Image File Format (可交换图像文件格式)，用于存储照片元数据 88888
ORM	Object-Relational Mapping (对象关系映射)，本项目中使用 SQLAlchemy 9
RESTful	Representational State Transfer，一种 Web API 设计风格
UI	User Interface (用户界面)
CRUD	Create, Read, Update, Delete (增、删、改、查)
MCP	Model Context Protocol (大模型上下文协议)，用于对话式检索 10
SDK	Software Development Kit (软件开发工具包)

1.5. 参考资料

序号	文档名称	文档编号/版本
1	实验:图片管理网站	assignment(2025秋冬) .pdf 11
2	Flask 官方文档	3.1
3	React 官方文档	v19.2

---

2. 项目介绍

2.1. 项目说明

- **项目名称**：图片管理网站
- **用户群**：需要存储、管理和检索个人照片的普通用户。
- 任务提出者：浙江大学 B/S 体系软件设计课程任课教师——胡晓军
- 开发者：葛芸曦 3230104150

2.2. 项目背景

在全球信息化和数字化的浪潮下，我们正处于一个“读图时代”。智能手机、专业相机等设备的普及，使得图像生成的速度和数量呈指数级增长。无论是个人生活记录（如旅行、家庭聚会）还是专业工作（如设计、摄影），图像都已成为信息传递和记忆存储的核心载体。为了应对数字影像资产碎片化和检索效率低下的挑战，我们利用现代Web技术和人工智能，构建一个功能完善、私有化部署的图片管理网站，**打通** 从上传、智能处理（EXIF解析、AI打标）、存储、多维度检索到友好展示和编辑**的全流程闭环。这不仅能帮助用户“盘活”沉睡的数字影像资产，极大提升管理和检索效率，同时也能确保用户对个人数据的绝对掌控权。

本项目还是《BS体系软件设计》课程的大作业，我们通过实践一个完整的 Web 项目，掌握前后端分离的开发技术。项目要求实现一个功能完善的图片管理网站，满足用户从上传、管理到检索的全流程需求 。

2.3. 需求概述

根据课程要求，系统需满足以下需求：

2.3.1 功能需求

1. **用户注册与登录**：

- 用户名、密码长度需大于等于6字节。
- Email 格式需验证 。
- 用户名和 Email 在系统中唯一。

2. **图片上传**：

- 支持 PC 和手机浏览器上传。
- 支持多种图片格式（JPEG, PNG, GIF, BMP, WebP） 。

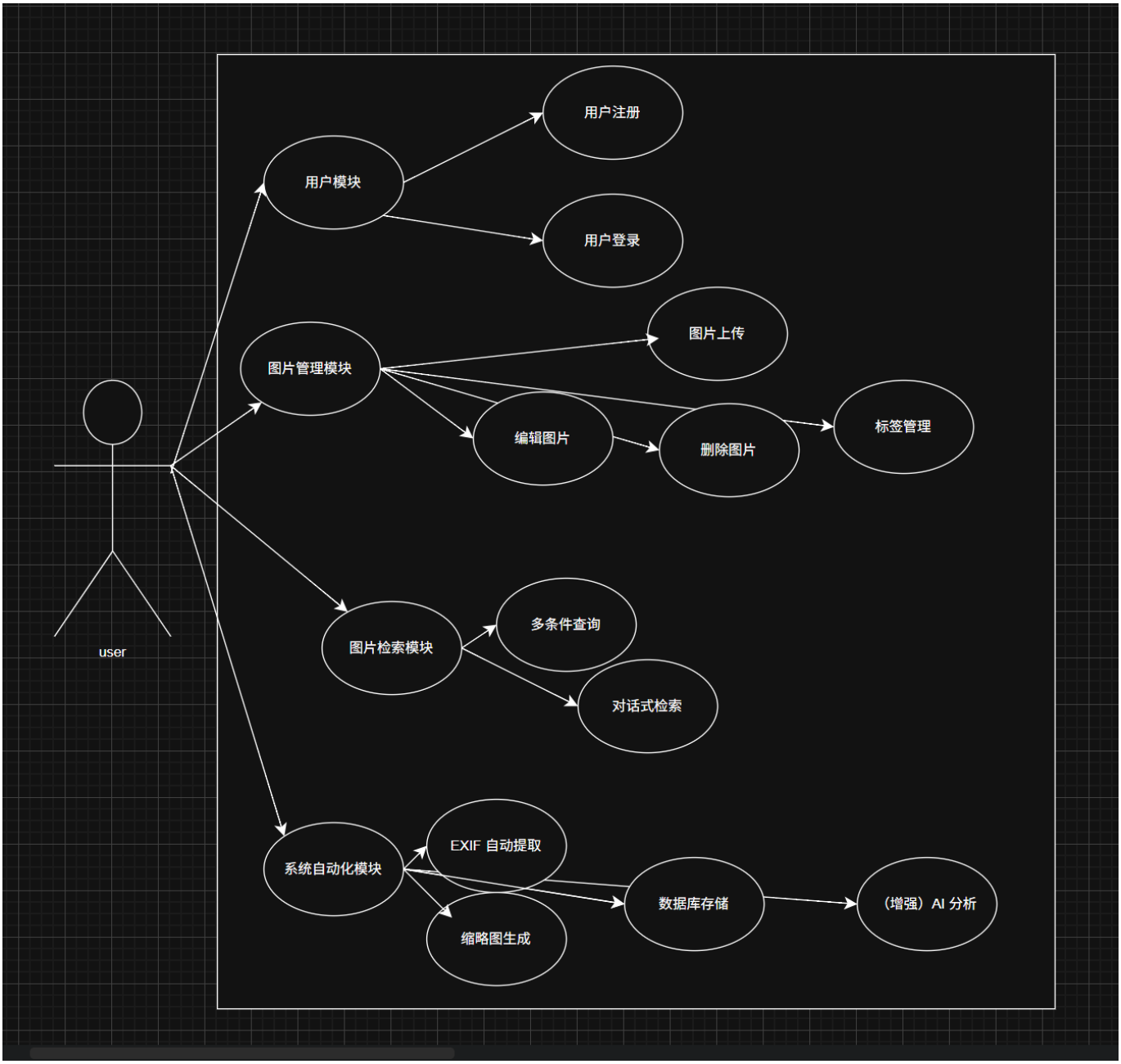
3. **EXIF 自动提取**：
    - 上传时自动提取拍摄时间、GPS 位置、相机型号、分辨率等。
4. **标签系统**：
    - 自动创建标签（基于 EXIF 的日期、相机型号等。
    - 支持用户添加、删除自定义标签。
5. **缩略图生成**：
    - 上传时自动生成缩略图 。

6. **数据库存储**：
    - 图片信息（元数据、路径、标签）保存在数据库中。
7. **查询界面**：
    - 支持多种条件查询：关键词（文件名）、标签、日期范围。
8. **删除功能**：
    - 用户可以删除自己上传的图片。
    - 删除时需同步删除服务器上的原图、缩略图及数据库记录 。
9. **编辑功能**
    - 对选定的图片提供简单的编辑功能，如裁剪、修改色调等
10. **移动端适配**：
    - 界面使用响应式设计，适配手机浏览器。
11. **增强功能（设计）**：
    - 提供 AI 分析图片生成标签（如风景、人物）的方案 。
    - 提供 MCP 接口，支持对话式检索的设计方案 。

2.3.2 用户类及其特征

用户类	描述
个人用户	系统的基础账户实体。拥有唯一的认证信息（用户名、密码、Email）和私有图片空间。可以上传、管理、检索自己的图片。这是系统的默认用户类型。

2.3.3 系统总用例图



2.3.3 性能需求

1. **响应时间**：页面加载和 API 响应应在 5秒内完成。
2. **图片加载**：缩略图优先加载，实现图片懒加载，优化用户体验。
3. **并发处理**：后端服务应能处理一定的并发请求（通过 Gunicorn 等 WSGI 服务器部署）。
4. **搜索效率**：数据库查询（特别是标签和 EXIF 搜索）应建立索引，保证大数据量下的查询速度。

2.3.4 安全需求（可进一步考虑实现）

1. **认证安全**：用户敏感操作（上传、删除）经过 JWT 认证。

2. **密码安全**：用户密码使用 `werkzeug.security` 进行加盐哈希（bcrypt）存储，不可明文存储。
3. **SQL 注入**：使用 SQLAlchemy ORM，避免直接拼接 SQL 字符串，防止 SQL 注入。
4. **文件上传**：
   - 严格验证文件类型白名单（`ALLOWED_EXTENSIONS`）。
   - 使用 `secure_filename` 处理文件名，并重命名为 UUID，防止路径遍历。
   - 限制上传文件大小（`MAX_CONTENT_LENGTH`）。
5. **CORS**：配置 CORS 策略，仅允许指定的前端域名访问 API。

2.4. 条件与限制

- 要求界面友好，提供必要的文档，包括设计文档和使用手册等其他文档。
- **运行环境**：需支持 Docker 容器化部署。
- 代码用 git 进行管理

---

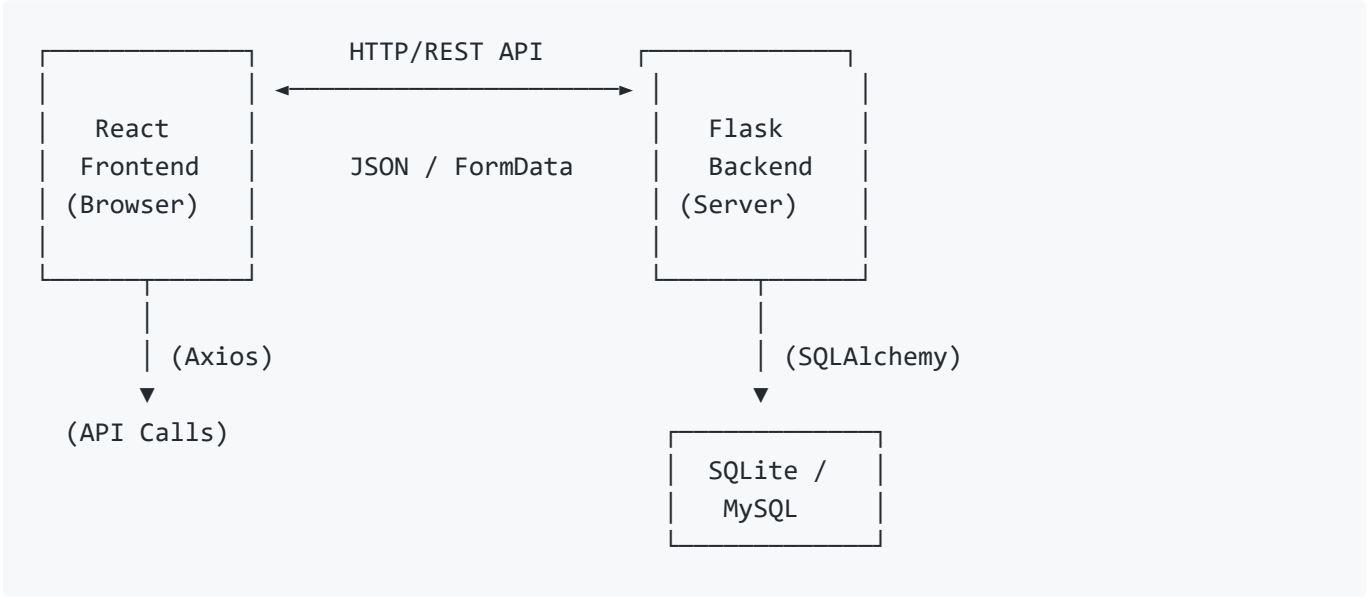
3. 总体设计

3.1. 基本设计概念和流程处理

本项目采用**前后端分离**的架构。

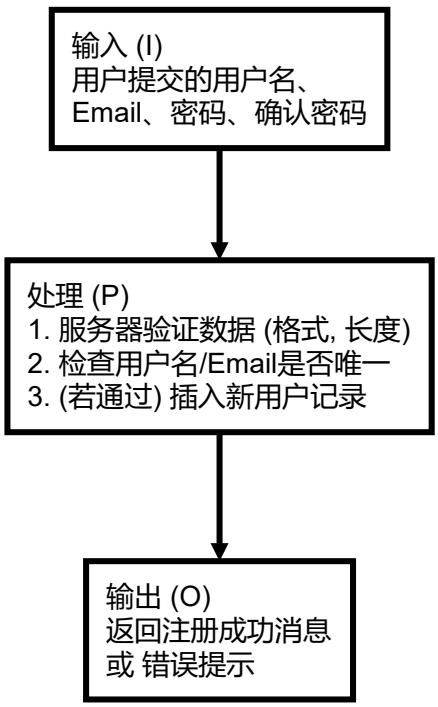
1. **前端 (Client)**：
   - 使用 React 和 Material-UI 构建的单页应用（SPA）。
   - 负责用户界面展示和交互逻辑。
   - 通过 Axios 调用后端 RESTful API。
   - 在本地（localStorage）存储 JWT 访问令牌和刷新令牌。
2. **后端 (Server)**：
   - 使用 Python Flask 构建的 RESTful API 服务器。
   - 负责业务逻辑处理、身份认证、图片处理和数据库交互。
3. **数据库 (Database)**：
   - 使用 MySQL。
   - 存储用户信息、图片元数据、标签及它们之间的关联
4. **文件存储 (Storage)**：
   - 上传的图片原图和缩略图存储在服务器的 `backend/uploads/` 和 `backend/thumbnails/` 目录中。

3.2 项目架构图：

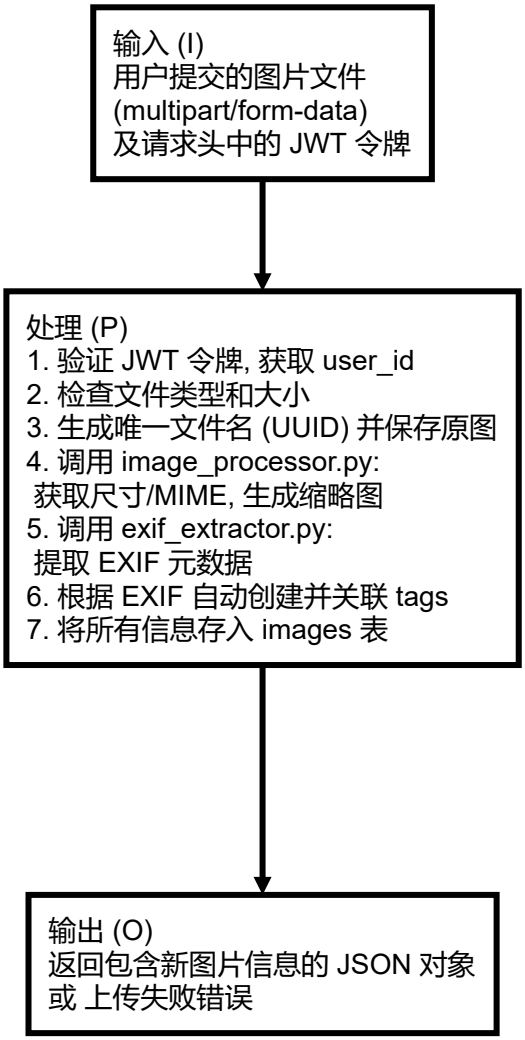


3.3 功能 IPO 图

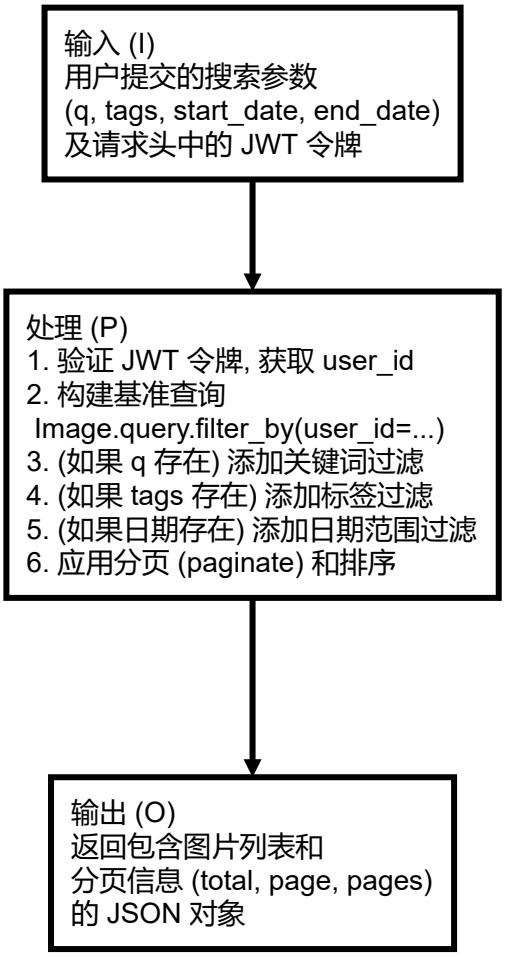
3.3.1. 用户注册



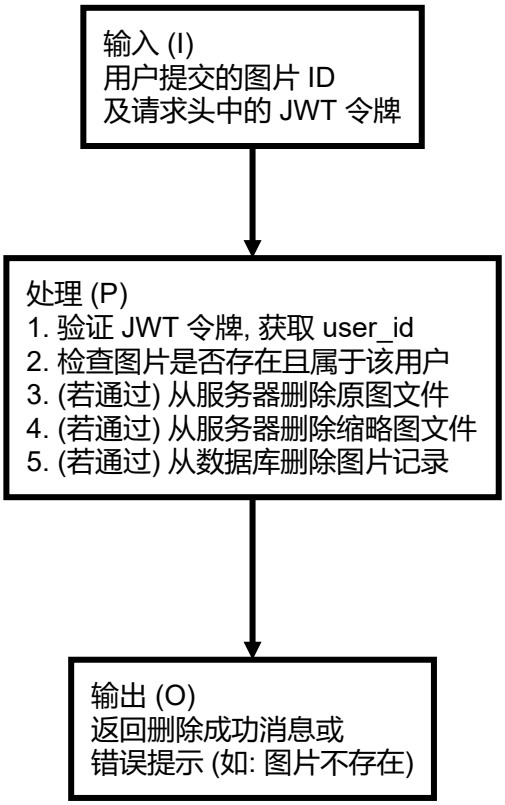
3.3.2. 图片上传


输入 (I)
用户提交的图片文件
(multipart/form-data)
及请求头中的 JWT 令牌
处理 (P)
1. 验证 JWT 令牌, 获取 user_id
2. 检查文件类型和大小
3. 生成唯一文件名 (UUID) 并保存原图
4. 调用 image_processor.py:
获取尺寸/MIME, 生成缩略图
5. 调用 exif_extractor.py:
提取 EXIF 元数据
6. 根据 EXIF 自动创建并关联 tags
7. 将所有信息存入 images 表
输出 (O)
返回包含新图片信息的 JSON 对象
或 上传失败错误

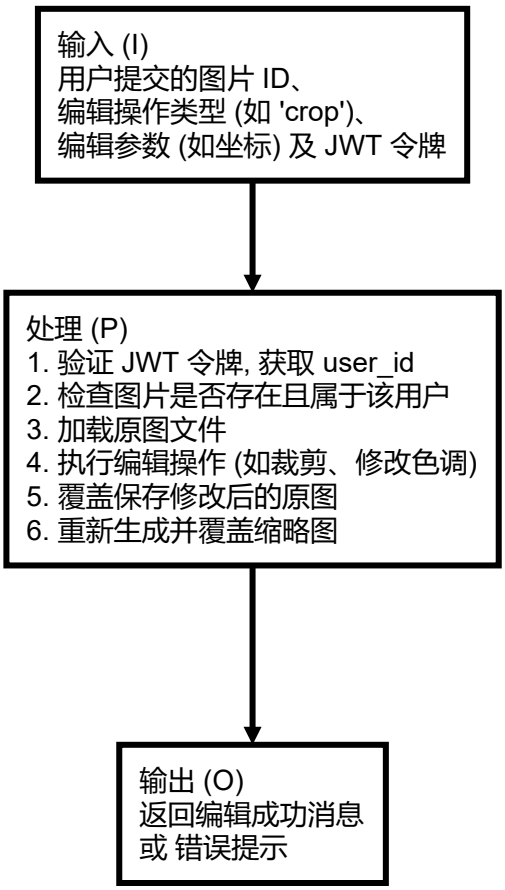
3.3.3. 多条件搜索


输入 (I)
用户提交的搜索参数
(q, tags, start_date, end_date)
及请求头中的 JWT 令牌
处理 (P)
1. 验证 JWT 令牌, 获取 user_id
2. 构建基准查询
Image.query.filter_by(user_id=...)
3. (如果 q 存在) 添加关键词过滤
4. (如果 tags 存在) 添加标签过滤
5. (如果日期存在) 添加日期范围过滤
6. 应用分页 (paginate) 和排序
输出 (O)
返回包含图片列表和
分页信息 (total, page, pages)
的 JSON 对象

3.3.4 删除图片

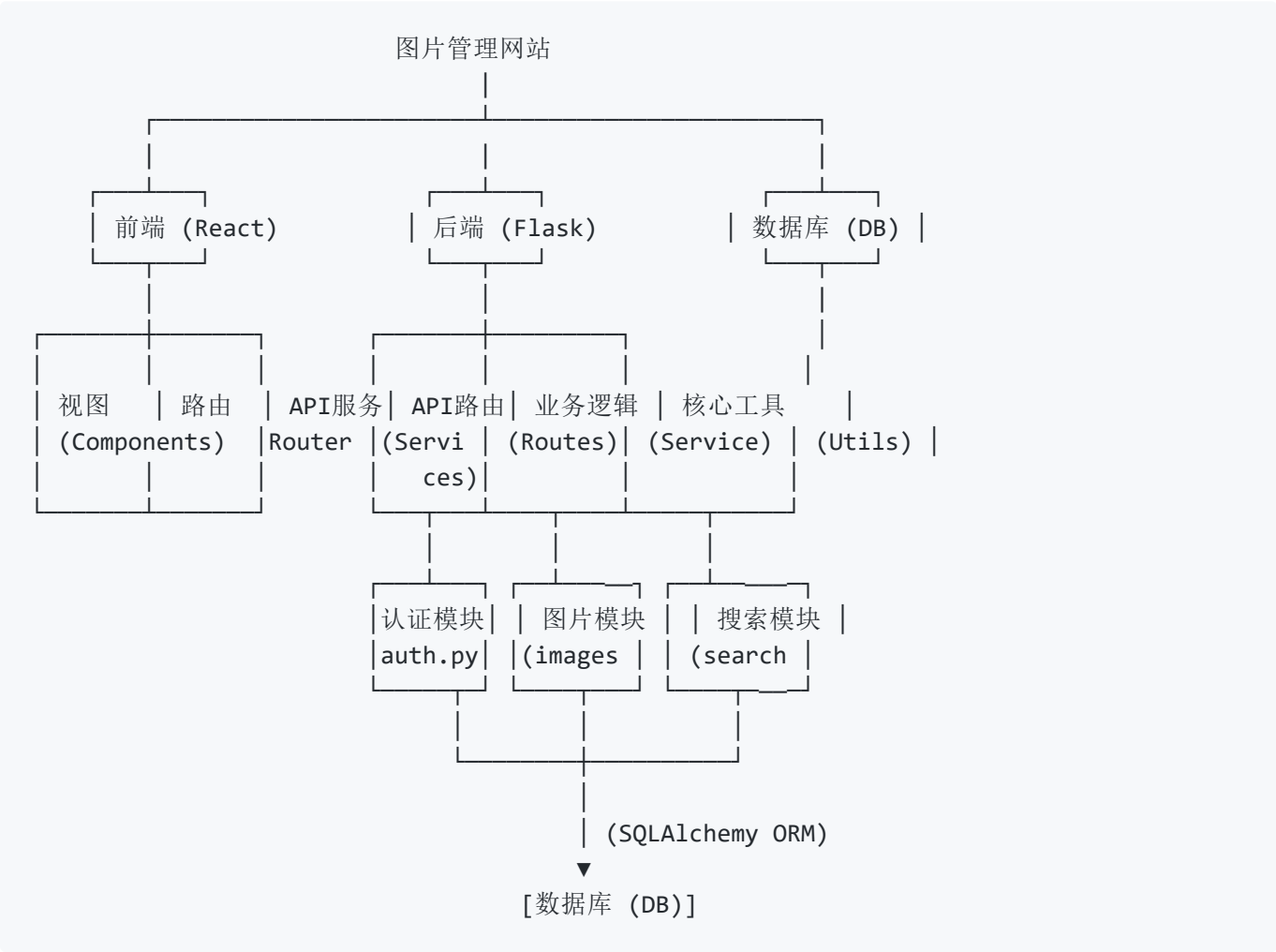

输入 (I)
用户提交的图片 ID
及请求头中的 JWT 令牌
处理 (P)
1. 验证 JWT 令牌, 获取 user_id
2. 检查图片是否存在且属于该用户
3. (若通过) 从服务器删除原图文件
4. (若通过) 从服务器删除缩略图文件
5. (若通过) 从数据库删除图片记录
输出 (O)
返回删除成功消息或
错误提示 (如: 图片不存在)

3.3.5 编辑图片



3.4. 系统功能模块图

系统按功能模块划分为以下几个主要部分：

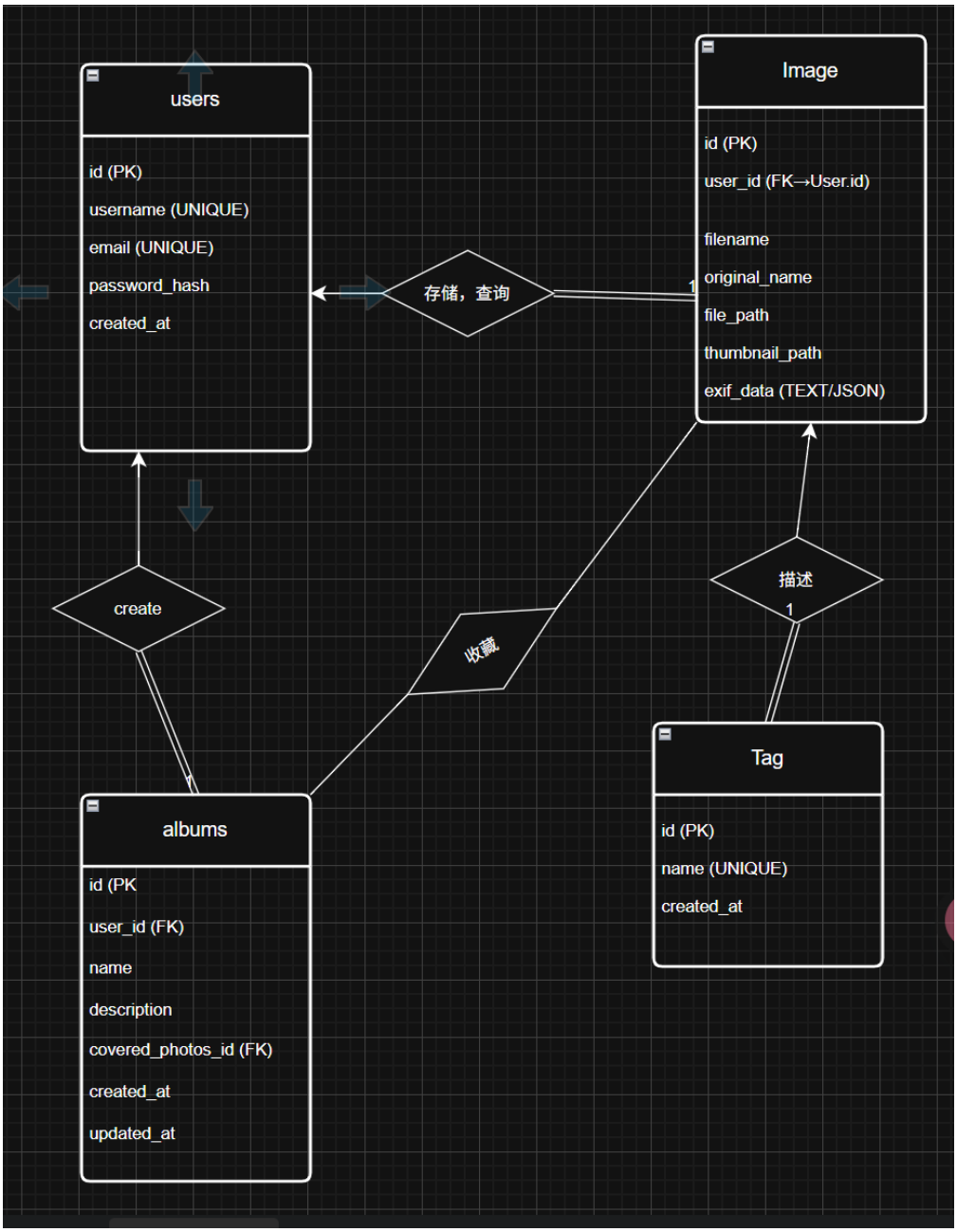


3.4. 技术介绍

- **后端**：
  - **Flask**：轻量级 Python Web 框架，用于构建 RESTful API。
  - Flask-SQLAlchemy：ORM，用于与数据库交互 。
  - **Flask-JWT-Extended**：处理 JWT 身份认证。
  - Pillow (PIL)：用于图片处理，如生成缩略图、获取尺寸。
  - piexif：用于读取和解析 EXIF 元数据 。
- **前端**：
  - **React**：用于构建用户界面的 JavaScript 库。
  - **React Router**：处理前端路由（SPA）。
  - **Material-UI (MUI)**：UI 组件库，提供响应式设计。
  - **Axios**：Promise-based 的 HTTP 客户端，用于 API GOGI。
  - **React Dropzone**：用于实现拖拽上传功能。
- **数据库**：
  - MySQL ： 生产环境推荐的关系型数据库。

- **部署**:
  - Docker： 用于容器化部署。
  - Gunicorn / Nginx： 生产环境的 WSGI 服务器和反向代理。

3.6. ER图 (数据模型)



3.7. 接口设计

3.7.1 内部接口

- `utils.image_processor.process_image(file_path)`：处理图片，返回 (width, height, size, mime) 。

- `utils.image_processor.generate_thumbnail(input, output)`: 生成缩略图。
- `utils.exif_extractor.extract_exif_data(image_path)`: 提取 EXIF 数据 。
- `utils.validators.validate_email(email)`: 验证邮箱格式。
- `models.User.check_password(password)`: 验证密码 。

3.7.2 外部接口 (RESTful API)

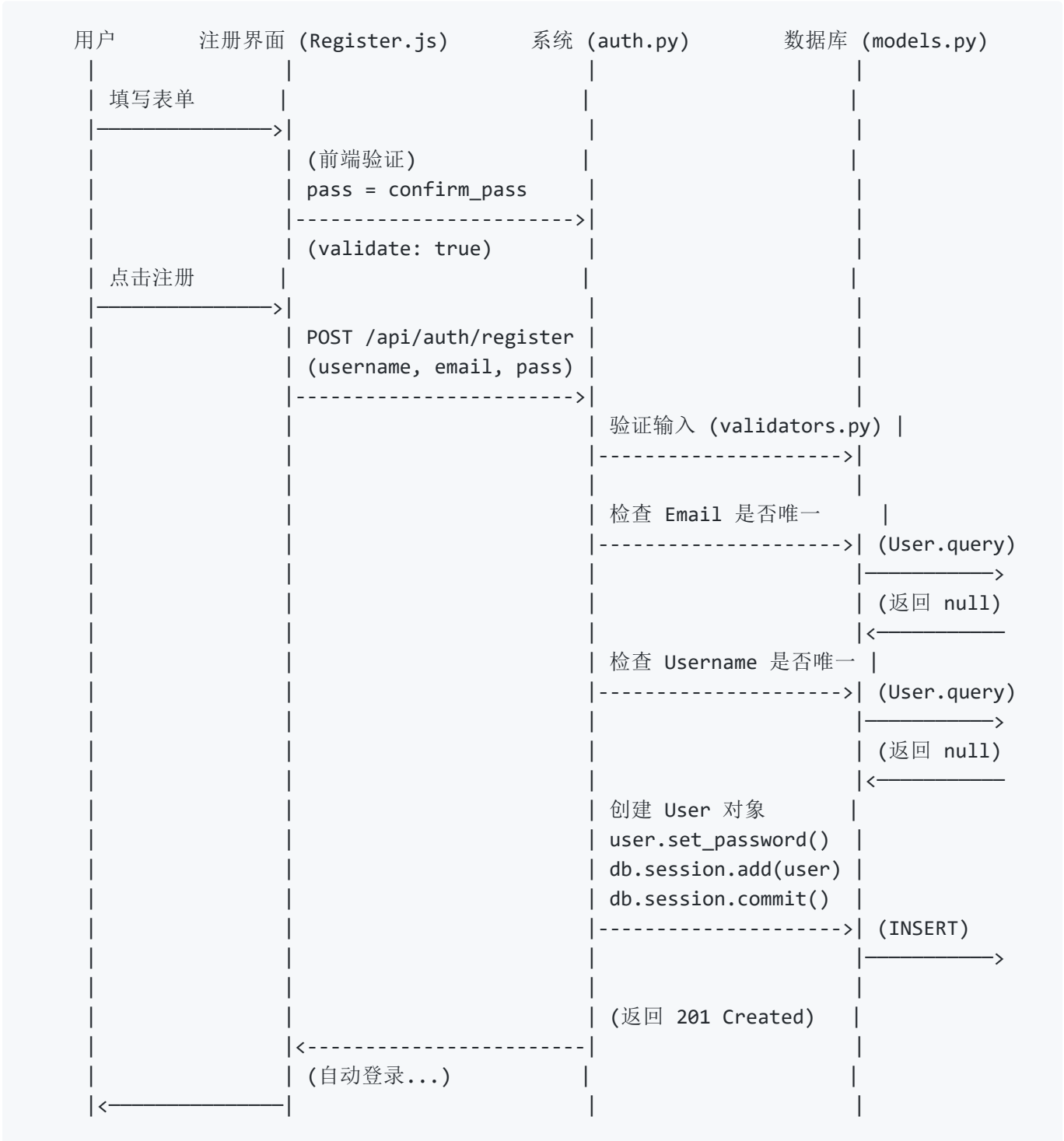
- **认证** (Auth):
  - `POST /api/auth/register`： 用户注册
  - `POST /api/auth/login`： 用户登录
  - `POST /api/auth/refresh`： 刷新 Access Token
  - `GET /api/auth/me`： 获取当前用户信息
- 图片 (Images):
  - `POST /api/images/upload`： 上传图片（需认证）
  - `GET /api/images`： 获取当前用户的图片列表（需认证）
  - `GET /api/images/<id>`： 获取图片详情（需认证）
  - `DELETE /api/images/<id>`： 删除图片（需认证）
  - `POST /api/images/<id>/tags`： 为图片添加标签（需认证）
  - `DELETE /api/images/<id>/tags/<tag_id>`： 移除图片标签（需认证）
  - `GET /api/images/file/<filename>`： 获取原图文件
  - `GET /api/images/thumbnail/<filename>`： 获取缩略图文件
- 搜索 (Search)：
  - `GET /api/search`： 综合搜索（支持 q, tags, start_date, end_date） （需认证）
  - `GET /api/search/tags`： 获取所有标签（需认证）
  - `GET /api/search/by-date`： 按日期搜索（需认证）
  - `GET /api/search/by-exif`： 按 EXIF 搜索（需认证）

---

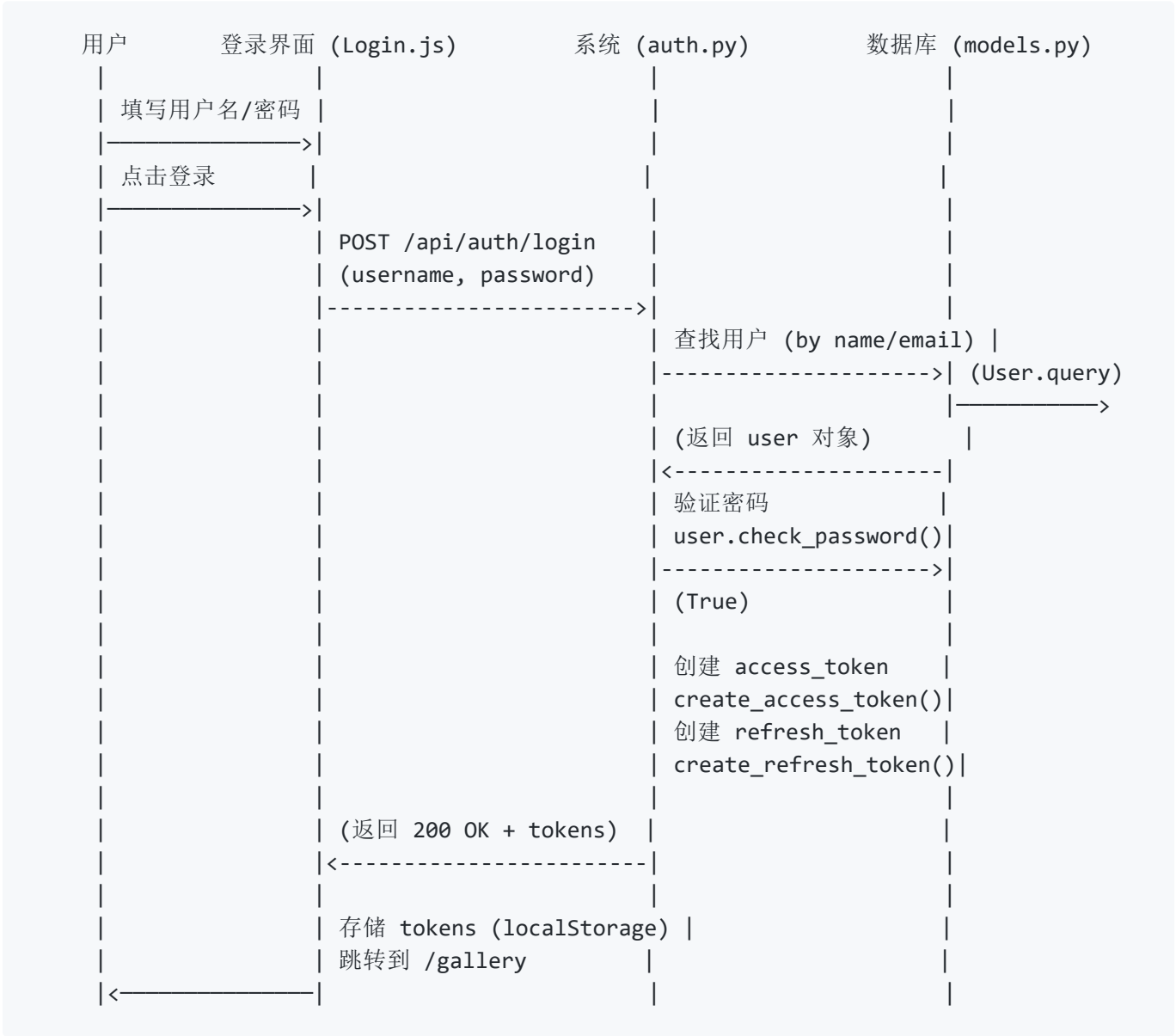
4. 详细设计

4.1. 顺序图

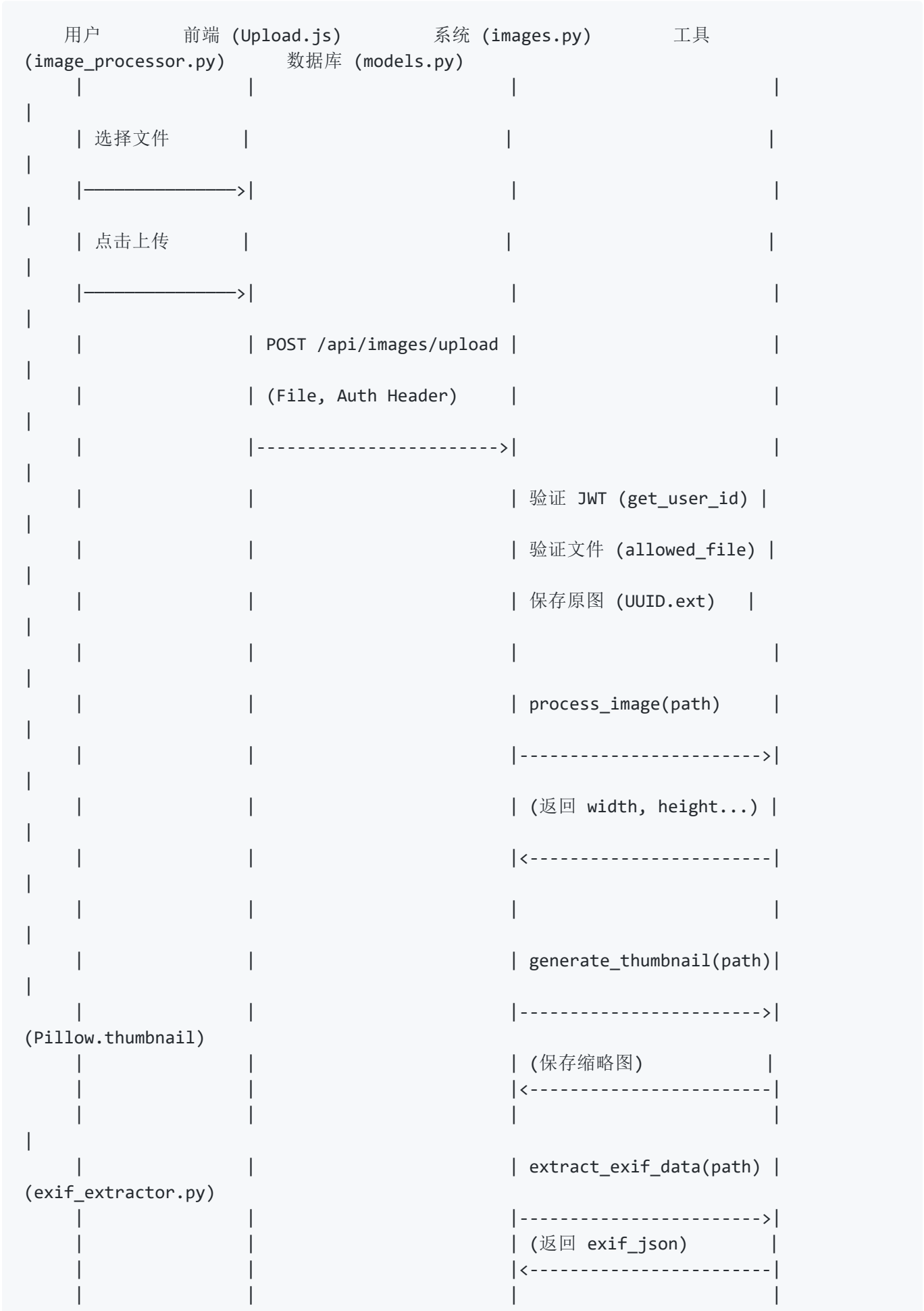
4.1.1 注册个人用户


用户
注册界面 (Register.js)
系统 (auth.py)
数据库 (models.py)
填写表单
(前端验证)
pass = confirm_pass
(validate: true)
点击注册
POST /api/auth/register
(username, email, pass)
验证输入 (validators.py)
检查 Email 是否唯一
(User.query)
(返回 null)
检查 Username 是否唯一
(User.query)
(返回 null)
创建 User 对象
user.set_password()
db.session.add(user)
db.session.commit()
(INSERT)
(返回 201 Created)
(自动登录...)

4.1.2 登录系统



4.1.3 上传图片（核心流程）

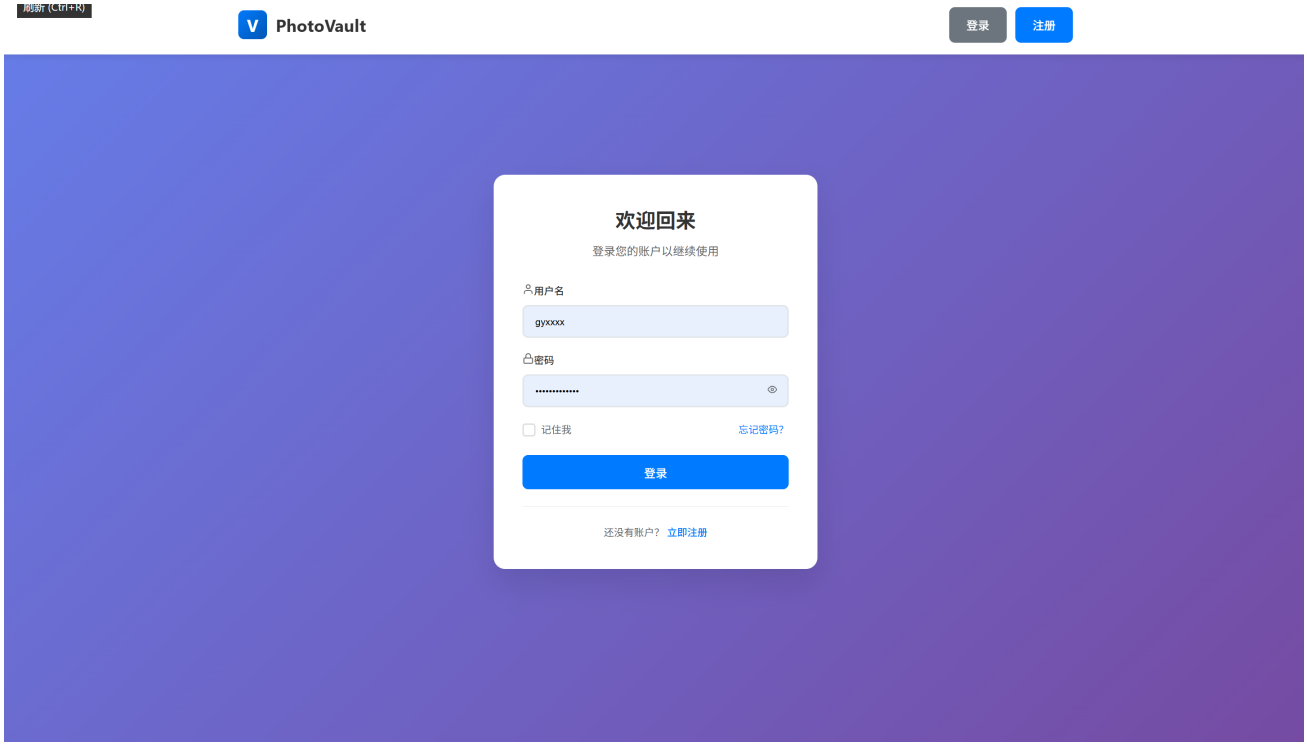

用户
前端 (Upload.js)
系统 (images.py)
工具 (image_processor.py)
数据库 (models.py)
选择文件
点击上传
POST /api/images/upload
(File, Auth Header)
验证 JWT (get_user_id)
验证文件 (allowed_file)
保存原图 (UUID.ext)
process_image(path)
(返回 width, height...)
generate_thumbnail(path)
(Pillow.thumbnail)
(保存缩略图)
extract_exif_data(path)
(exif_extractor.py)
(返回 exif_json)

```
|
     |                |                          | 创建 Image 对象        |
     |                |                          | 调用API自动创建 Tags         |
     |                |                          | db.session.add(image) |
     |                |                          | db.session.commit()   |
     |                |                          |---------------------->| (INSERT)
     |                |                          |                        |
————————>
     |                |                          |                        |
     |                | (返回 201 Created)       |                        |
     |                |<-------------------------|                        |
     | 刷新图库        |                          |                        |
     |<———————————————|                          |                        |
```

---

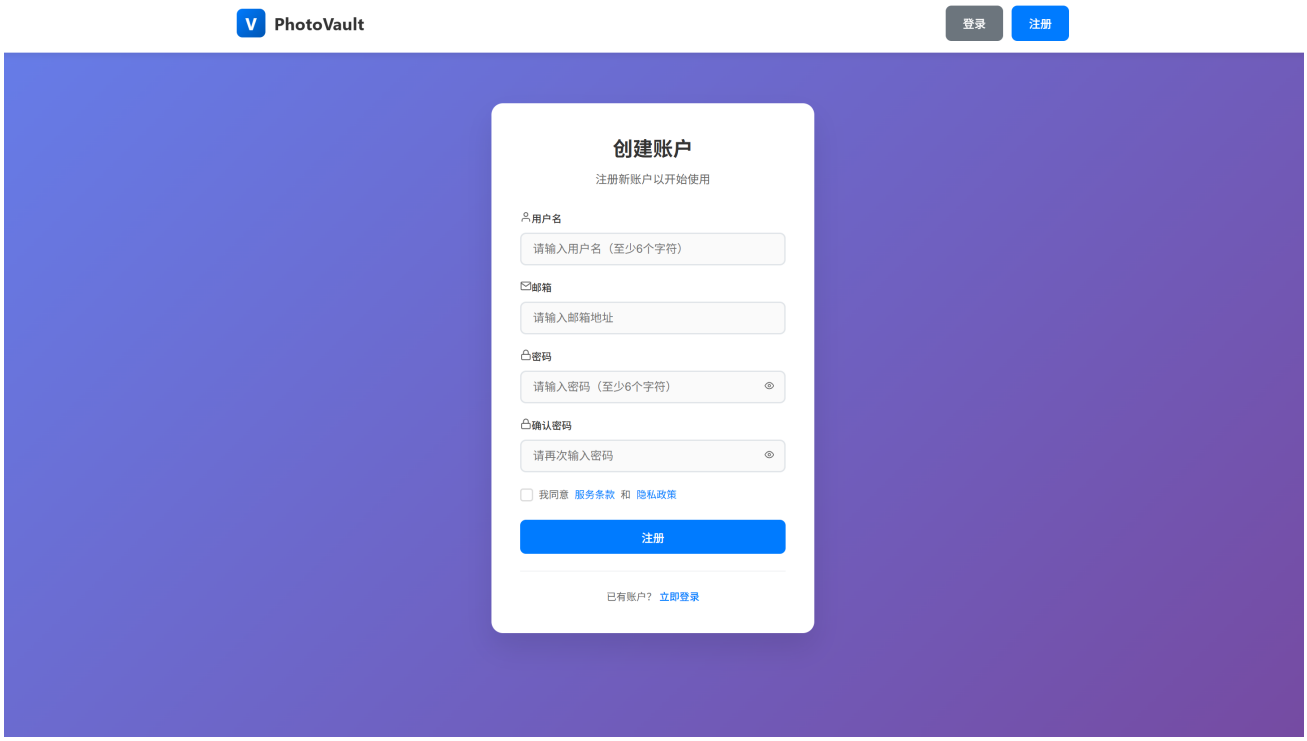
5. 用户界面

- **登录界面**：



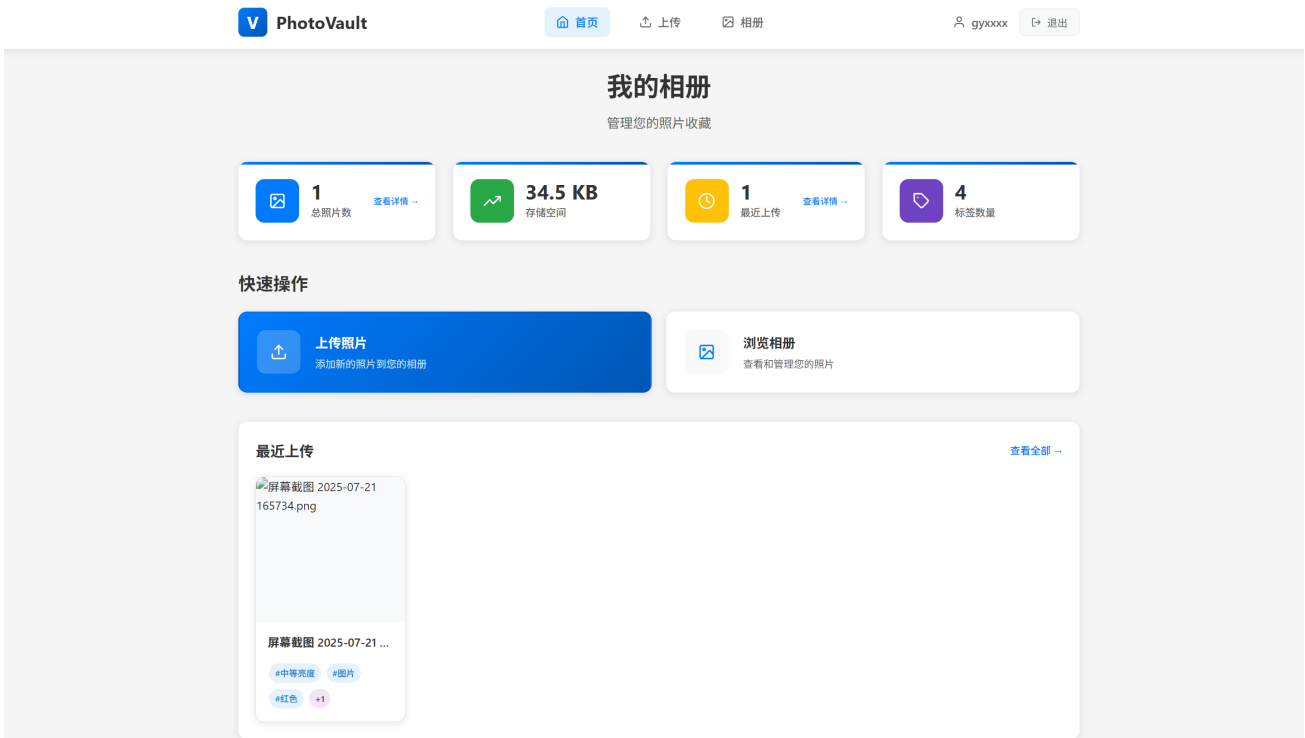
  - 提供“用户名或邮箱”和“密码”输入框。
  - 包含表单前端验证逻辑（如密码长度）。
  - 错误时显示 `<Alert>` 提示信息。
  - 登录成功后，调用 `localStorage.setItem` 存储 `access_token` 和 `refresh_token`，并跳转到图库页（`/gallery`）。
  - 提供“立即注册”的链接。

- **注册界面**：



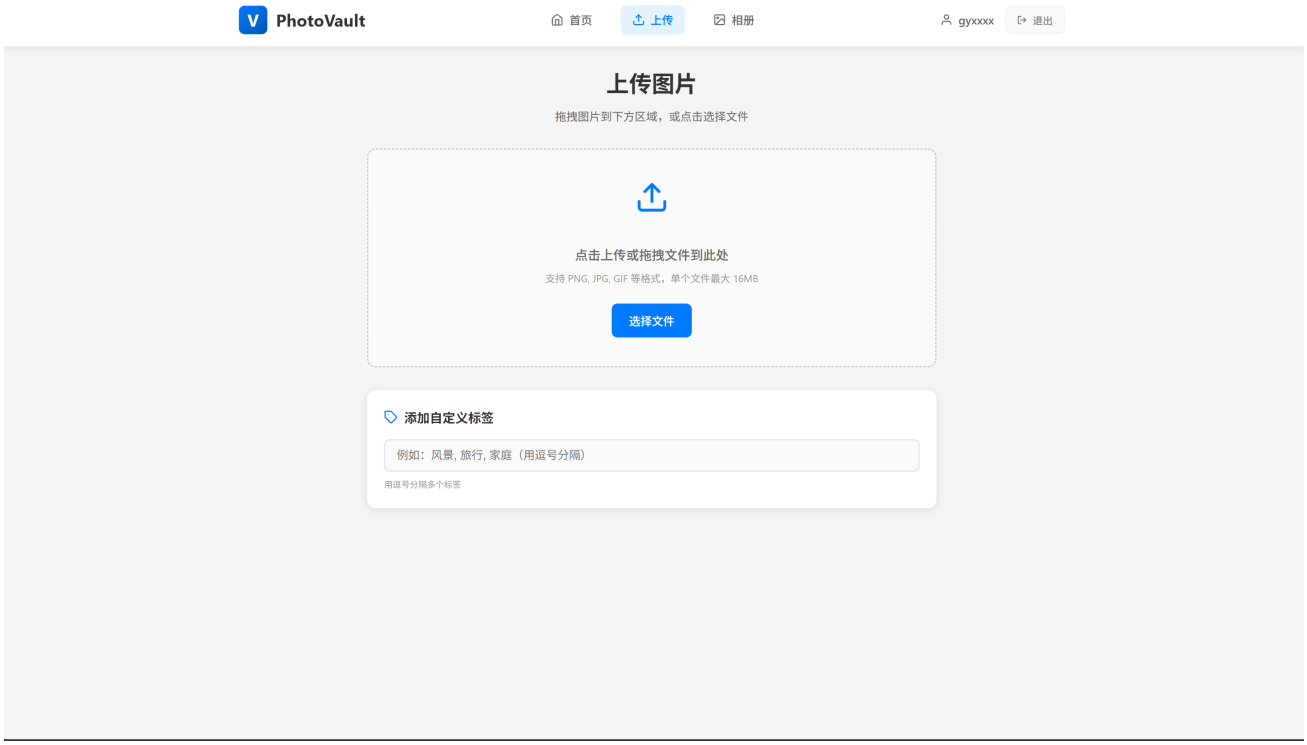
  - 提供“用户名”、“邮箱”、“密码”、“确认密码”输入框。
  - 包含详细的前端验证，如用户名/密码长度（>=6）、邮箱格式、两次密码一致性 。
  - 使用 `helperText` 实时提示错误。
  - 注册成功后，自动调用登录 API 并跳转到图库页。

- **图库界面**：



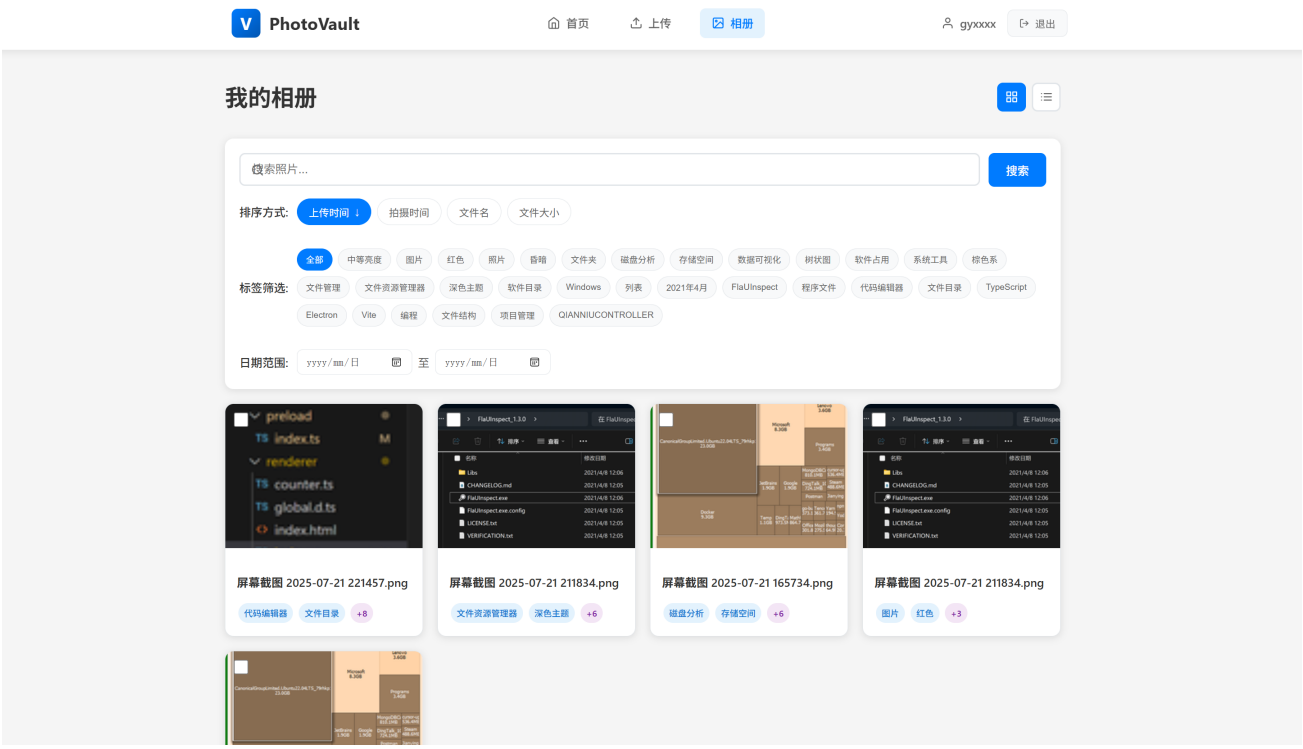
- 提供上传图片和浏览相册两个按钮
- 显示图片缩略图，可以直接点击查看图片详情。
- 在页首提供不同页面的跳转还有用户的登出。

- **上传界面**：



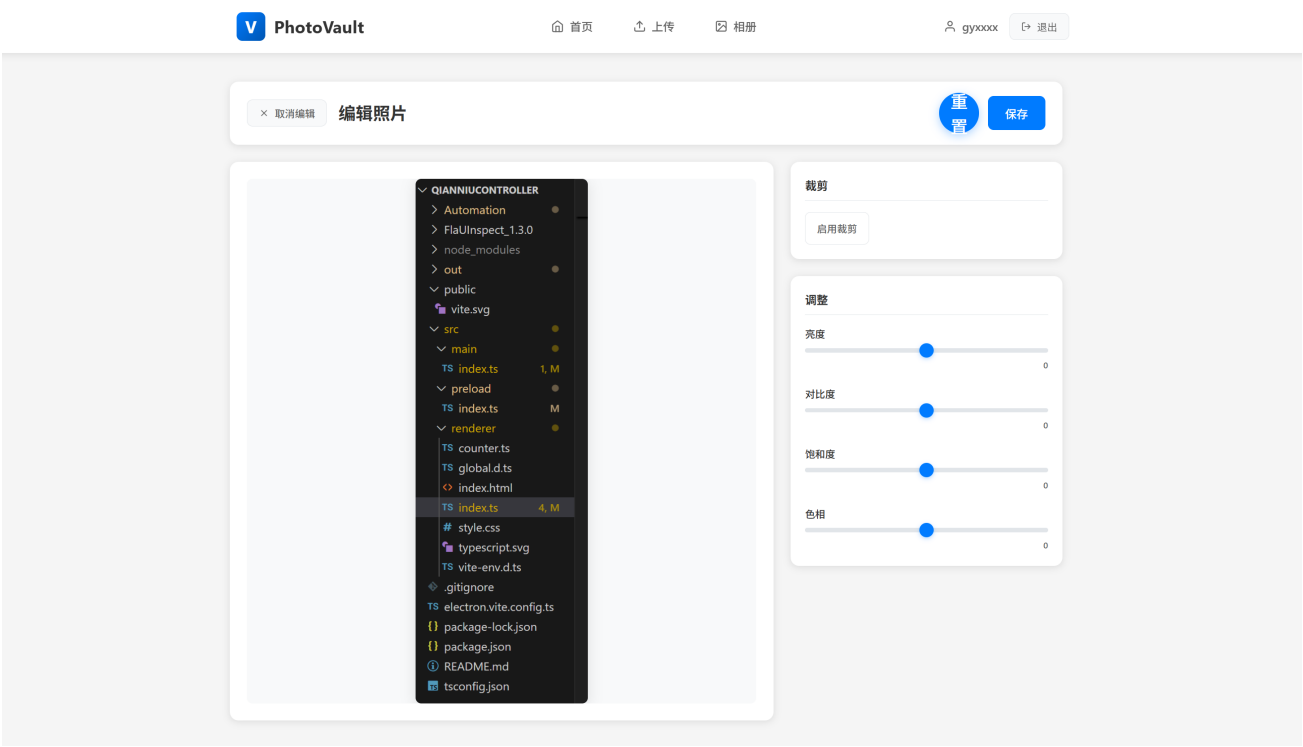
- 有一个拖拽上传区域，点击选择文件可以打开文件资源管理器选择要上传的图片。
- 显示上传进度。
- 可以添加自定义标签。

- **搜索界面**：



  - 有一个搜索框用于文件名字搜索
  - 下面可以选择标签限定搜索范围。并且选择排序方式，有上传时间拍摄时间文件名文件大小。
  - 也包含日期范围选择器。
  - 搜索结果在下方网格中显示，点击可以进入图片编辑详情页。

- 图片编辑页面：



- 图片在正中间，右边可以进行简单的编辑工作比如裁剪，亮度对比度饱和度色相的调整
- 上方选择重置编辑或者保存编辑或者取消编辑状态

---

6. 数据库设计

6.1. 概念结构设计

本项目的核心实体及其关系如下：

1. **User (用户)**：存储用户信息。
2. **Image (图片)**：存储图片元数据，是核心实体。
3. **Tag (标签)**：存储标签信息。

关系：

- 一个 `User` 可以拥有多张 `Image` (1:N)。
- 一张 `Image` 只能属于一个 `User` (N:1)。
- 一张 `Image` 可以拥有多个 `Tag` (N:M)。
- 一个 `Tag` 可以赋给多张 `Image` (N:M)。

6.2. 逻辑结构设计

表 1: users (用户表)

属性名	字段名称	数据类型	长度	备注
用户 ID	id	INT		主键, 自增
用户名	username	VARCHAR	50	不空, 唯一
邮箱	email	VARCHAR	100	不空, 唯一
密码哈希	password_hash	VARCHAR	255	不空
创建时间	created_at	TIMESTAMP		默认当前时间
更新时间	updated_at	TIMESTAMP		默认当前时间, 更新时刷新

表 2: photos (图片表)

属性名	字段名称	数据类型	长度	备注
图片 ID	id	INT		主键, 自增
用户 ID	user_id	INT		不空, 外键 (users.id), 级联删除
文件名	filename	VARCHAR	255	不空 (UUID)
原始文件名	original_filename	VARCHAR	255	不空
文件路径	file_path	VARCHAR	500	不空
缩略图路径	thumbnail_path	VARCHAR	500	
文件大小	file_size	BIGINT		不空 (Bytes)
MIME 类型	mime_type	VARCHAR	100	不空
宽度	width	INT		
高度	height	INT		
相机制造商	camera_make	VARCHAR	100	
相机型号	camera_model	VARCHAR	100	
拍摄时间	taken_at	DATETIME		
拍摄纬度	latitude	DECIMAL	10, 8	
拍摄经度	longitude	DECIMAL	11, 8	
地点名称	location_name	VARCHAR	200	
创建时间	created_at	TIMESTAMP		默认当前时间
更新时间	updated_at	TIMESTAMP		默认当前时间, 更新时刷新

表 3: tags (标签表)

属性名	字段名称	数据类型	长度	备注
标签 ID	id	INT		主键, 自增
标签名	name	VARCHAR	100	不空, 唯一
标签类型	type	ENUM		'auto' 或 'custom', 默认 'custom'
创建时间	created_at	TIMESTAMP		默认当前时间

表 4: photo_tags (图片标签关联表)

属性名	字段名称	数据类型	长度	备注
关联 ID	id	INT		主键, 自增
图片 ID	photo_id	INT		不空, 外键 (photos.id), 级联删除
标签 ID	tag_id	INT		不空, 外键 (tags.id), 级联删除
创建时间	created_at	TIMESTAMP		默认当前时间
(约束)				(photo_id, tag_id) 组合唯一

表 5: albums (相册表)

属性名	字段名称	数据类型	长度	备注
相册 ID	id	INT		主键, 自增
用户 ID	user_id	INT		不空, 外键 (users.id), 级联删除
相册名称	name	VARCHAR	200	不空
描述	description	TEXT		
封面图ID	cover_photo_id	INT		外键 (photos.id), 设为 NULL
创建时间	created_at	TIMESTAMP		默认当前时间
更新时间	updated_at	TIMESTAMP		默认当前时间, 更新时刷新

表 6: album_photos (相册图片关联表)

属性名	字段名称	数据类型	长度	备注
关联 ID	id	INT		主键, 自增
相册 ID	album_id	INT		不空, 外键 (albums.id), 级联删除
图片 ID	photo_id	INT		不空, 外键 (photos.id), 级联删除
排序	sort_order	INT		默认 0
创建时间	created_at	TIMESTAMP		默认当前时间
(约束)				(album_id, photo_id) 组合唯一

7. 运行设计

用户使用本系统的典型流程如下：

1. **注册**：新用户访问前端页面，点击“注册”，填写用户名（>=6 字节）、邮箱和密码（>=6 字节），提交注册 。
2. **登录**：已注册用户访问“登录”页，输入用户名（或邮箱）和密码。系统验证通过后，前端保存 JWT 令牌，并跳转到图库主页。
3. **上传**：用户进入“上传”页，通过拖拽或点击选择图片。图片上传成功后，系统自动处理（EXIF、缩略图）并存入数据库 ，同时会调用gemini-2.0-flase的API解析图片自动生成5—10个标签。
4. **浏览**：用户在“图库”页浏览自己上传的所有图片缩略图。
5. **搜索**：用户在“搜索”页，通过关键词、标签或日期范围筛选图片 。
6. **管理**：用户在图库或图片详情页，可以为图片添加或删除自定义标签，或选择删除图片
7. **令牌刷新**：如果用户长时间操作，`access_token` 过期（401 错误），`api.js` 中的响应拦截器会自动使用 `refresh_token` 换取新的 `access_token`，用户无感知。
8. **退出**：用户点击“退出”，前端清除 `localStorage` 中的令牌，跳转回登录页。

8. 系统出错设计

8.1. 出错信息

系统输出信息的形式	含义	处理办法或相应对策
前端		
"用户名至少6个字符"	前端表单验证失败	提示用户修改输入，在满足条件前禁用提交按钮。
"两次输入的密码不一致"	前端表单验证失败	提示用户修改输入。
"登录失败，请重试"	API 返回了未知错误	提示用户检查网络或稍后重试。
后端 (API)		
400 Bad Request	请求数据不合法（如“邮箱格式不正确”、“密码必须至少6个字符”）	前端应捕获此错误并展示给用户。
400 Bad Request	“没有文件上传”或“不支持的文件类型”	前端应在上传前进行检查，或在失败后提示用户。
401 Unauthorized	JWT 令牌无效或过期	1. (未提供令牌) 前端跳转到登录页。2. (令牌过期) `api.js` 拦截器尝试刷新令牌；刷新失败则跳转登录页。
404 Not Found	资源不存在（如 `GET /api/images/999` ）	提示用户“图片不存在”或“资源未找到”。
500 Internal Server Error	服务器内部异常（如数据库连接失败、图片处理失败）	提示用户“服务器繁忙，请稍后重试”。后端需记录详细日志。

8.2. 补救措施

1. **系统恢复 (Token 刷新)**：
    - 当 API 返回 401 状态码时，拦截器会检查是否存在 `refresh_token` 。
    - 如果存在，它会自动调用 `/api/auth/refresh` 接口获取新的 `access_token` ，并重试失败的请求，实现无感知的会话恢复。
    - 如果刷新失败（ `refresh_token` 也过期），则清除所有 token 并强制用户返回登录页。
2. **定时备份 (部署)**：

- **数据库备份**：使用 `mysqldump` 定期（如每日）备份 MySQL 数据库。对于 SQLite，可直接复制数据库文件 。
- **文件备份**：定期（如每日）使用 `tar` 或 `rsync` 备份 `uploads/` 和 `thumbnails/` 目录 。

3. **系统维护设计 (日志)**：
   - 应配置 Flask 的日志级别，将应用错误和访问日志记录到文件。
   - 定期监控日志，使用 `logrotate` 进行日志轮转。
   - 定期监控磁盘空间，特别是 `uploads/` 目录。