

浙江大学

课程名称: BS体系软件设计

实验名称: 图片管理网站

姓 名: 胡航溢

学 院: 计算机科学与技术学院

系: 计算机科学与技术系

专 业: 计算机科学与技术

学 号: 3230102920

指导教师: 胡晓军

报告日期: 2025年12月21日

浙江大学实验报告

课程名称: BS体系软件设计 实验类型: 软件

实验项目名称: 图片管理网站

学生姓名: 胡航溢 专业: 计算机科学与技术 学号: 3230102920

同组学生姓名: _____ 指导老师: 胡晓军

实验地点: 曹光彪西-304 实验日期: 2025年12月21日

摘要

本实验基于 React + Vite (前端) 与 Flask (后端) 实现了一个图片管理与展示系统。系统支持用户注册与登录、图片上传、EXIF 信息解析、缩略图生成、标签管理、图片展示与基础编辑功能；后端以 SQLite 存储元数据，并通过 JWT 实现鉴权，提供检索与管理类 API。本文概述系统架构与关键实现，说明主要设计决策、部署方式与测试结果，并提出后续优化方向。

关键词: 图片管理、EXIF、缩略图、JWT、React、Flask

1 引言

本实验旨在实现一个前后端分离的图片管理网站原型，覆盖用户认证、图片上传与处理、标签管理、检索与展示等核心功能，并对关键模块的设计与实现进行说明与测试验证。通过该原型验证图像元数据解析（包括 EXIF）、缩略图生成、标签推荐及前后端协作流程的可行性，为后续扩展（如更强的语义检索、分布式存储与高并发支持）提供实现依据。

设计与实现参考文档见 [Design.md](#) 与项目说明 [README.md](#)。

2 系统概述与架构

- 技术栈：前端 React + Vite + Tailwind（样式），后端 Flask + SQLAlchemy，图像处理使用 Pillow 与 exifread，数据库为 SQLite。
- 部署方式：支持本地开发与 Docker Compose 一键部署（见 [docker-compose.yml](#)）。
- 总体结构参见设计文档（模块划分、数据表、API 设计）：[Design.md](#)。

系统组件：

- 前端（SPA）：路由、上传面板、画廊、图片详情、用户中心（代码位置：[frontend/](#)）。
- 后端（REST API）：鉴权、图片处理、标签管理、推荐接口（入口：[backend/app.py](#)；路由：[backend/routes/](#)）。

3 关键实现

3.1 前端实现（简述）

- 使用 React 组件化实现：[UploadPanel.jsx](#)，[GalleryGrid.jsx](#)，[ImageCard.jsx](#)，[ImageCarousel.jsx](#) 等组件负责 UI 与交互。

[UploadPanel.jsx](#)（核心）

```
// 选择文件 / 支持拖拽
function UploadPanel({ onSelect }) {
  const handleFile = f => { if (f?.type?.startsWith('image/')) onSelect(f) }
  return (<div onDrop={e=>
    e.preventDefault(); handleFile(e.dataTransfer.files[0])}> onClick={()
      => document.getElementById('fileInput').click()}>
    <input id="fileInput" type="file" accept="image/*" style={{display:'none'}}/>
    onChange={e=>handleFile(e.target.files[0])} />
    <p>Click or Drag image here</p>
  </div>)
}
```

[GalleryGrid.jsx](#)（核心）

```
// 简化：把图片数组映射为 ImageCard 列表
export default function GalleryGrid({ images, onChange }){
  if(!images?.length) return <p>暂无图片</p>
  return <div className="gallery-grid">{images.map(i=> <ImageCard key={i.id}
    image={i} onChange={onChange} />)}</div>
}
```

ImageCard.jsx (核心逻辑)

```
// 渲染缩略图：点击增加 view；点击心形 toggle like
function ImageCard({image,onChange}){
  const handleView=async()=>{const r=await viewImage(image.id); onChange?.()}
  const handleLike=async e=>{e.stopPropagation(); const r=await
  toggleLike(image.id); onChange?.()}
  return (<div onClick={handleView}><img src=
  {'http://localhost:5000${image.thumbnail_url}' } alt="" /><div>#
  {image.primary_tag}</div><button onClick={handleLike}>❤</button></div>)
}
```

ImageCarousel.jsx (核心逻辑)

```
// 自动轮播 + 基本无缝处理（略去细节）
function ImageCarousel({images=[]}){
  useEffect(()=>{if(images.length>1){const
  t=setInterval(()=>setIdx(i=>i+1),3000);return ()=>clearInterval(t)}},[images])
  return (<div className="carousel">{images.map(img=> <img src=
  {'http://localhost:5000${img.url}' } />)}</div>)
}
```

- 与后端交互通过封装的 API 客户端（ frontend/src/api/api.js 、 auth.js 、 image.js ）； 登录后将 access_token 存为本地 Token，用于后续请求头 Authorization: Bearer <token> 。

frontend/src/api/api.js

```
// 创建 axios 实例并自动附带本地 token
import axios from 'axios'

const api = axios.create({ baseURL: 'http://localhost:5000/api' })

api.interceptors.request.use(config => {
  const token = localStorage.getItem('token')
  if (token) config.headers.Authorization = `Bearer ${token}`
  return config
})

export default api
```

frontend/src/api/auth.js

```
// 登录/注册/登出（示例）
import api from './api'

export async function login(username, password){
  const res = await api.post('/login', { username, password })
  const token = res.data.access_token
  localStorage.setItem('token', token)
  return res.data
}

export async function register(username, email, password){
  return api.post('/register', { username, email, password })
}

export function logout(){
  localStorage.removeItem('token')
}
```

frontend/src/api/image.js

```
// 图片相关接口：分析/上传/列表/浏览/点赞
import api from './api'
```

```

export async function analyzeImage(file){
    const fd = new FormData(); fd.append('file', file)
    return api.post('/images/analyze', fd)
}

export async function uploadImage(file, tags=[]){
    const fd = new FormData(); fd.append('file', file); fd.append('tags',
JSON.stringify(tags))
    return api.post('/images', fd)
}

export async function fetchImages(params){
    return api.get('/images', { params })
}

export async function viewImage(id){
    return api.get(`/images/${id}/view`)
}

export async function toggleLike(id){
    return api.post(`/images/${id}/like`)
}

```

- 开发运行：进入 `frontend` 目录，`npm install` 后 `npm run dev`（详见 [frontend/README.md](#)）。
- 前端分层（Pages + Components）：我们把前端按页面（Page）与组件（Component）分层实现。Pages（如 `Home.jsx`、`Gallery.jsx`、`Upload.jsx`、`Edit.jsx`、`Login.jsx`、`Register.jsx`、`UserCenter.jsx`）负责路由、数据请求与页面级状态管理，组件（如 `UploadPanel.jsx`、`ImageCard.jsx`、`GalleryGrid.jsx`、`ImageCarousel.jsx`、`TagSelector.jsx`）负责可复用的 UI 与局部交互。这一层次化设计使 Page 负责业务流程（上传/分析/展示/编辑），组件负责展示与交互，便于维护与复用。

3.2 后端实现（简述）

- 入口：[backend/app.py](#)：Flask 应用初始化、CORS、JWT、静态图片路由 `/uploads/<path>`，并注册蓝图。
- 路由实现：[backend/routes/image_routes.py](#)：包含图片上传（POST `/api/images`）、图片分析

(POST /api/images/analyze)、列表 (GET /api/images)、删除、编辑、点赞、浏览计数与推荐等逻辑。

- 图片上传流程要点：

- 校验文件类型 (允许 png/jpg/jpeg/gif)，保存原图到 backend/data/uploads/original/，生成缩略图到 backend/data/uploads/thumbs/。
- 使用 exifread 提取 EXIF (时间、GPS、设备信息)，并使用 utils 中的工具 (如 utils/thumbs.py, utils/exif_tag_helper.py) 生成推荐标签。
- 元数据写入 SQLite (数据库初始化在 backend/database.py)。
- 部分 AI 标签示例：代码中提供了调用百度图像识别的示例 (baidu_image_recognition)，以演示如何结合 AI 服务生成标签 (需替换合法 API Key)。

下面给出后端各功能的简短示例伪代码 (核心流程)，并标注实现文件：

(路由/文件)：backend/routes/image_routes.py — 上传 (upload_image) 核心流程

```
@jwt_required()
def upload_image():
    file = request.files['file']
    if not allowed_file(file.filename): return 400

    # 在 DB 建占位记录，取得 image.id
    image = Image(user_id=..., filename=' ', ...)
    db.add(image); db.commit(); db.refresh(image)

    # 保存原图并生成缩略图（调用 utils/thumbs.py）
    file.save(original_path)
    create_thumbnail(src_path=original_path, dst_path=thumb_path)

    # 提取 EXIF / 绑定 tags
    exif_time = extract_exif_time(original_path)
    for name in tag_names:
        tag = db.query(Tag).filter(Tag.name==name).first()
        if not tag:
            tag = Tag(name=name); db.add(tag); db.flush()
        image.tags.append(tag)

    db.commit()
    return {...}
```

(路由/文件) : backend/routes/image_routes.py — 临时分析 (analyze_image_exif)

```
def analyze_image_exif():
    tmp.write(file.stream.read())
    exif_time = extract_exif_time(tmp.name)
    gps = extract_exif_gps(tmp.name)
    ai_labels = baidu_image_recognition(tmp.name) # 可选
    suggested = generate_exif_tags(exif_time, width, height, gps_info, device_info,
ai_labels)
    return { 'exif': {...}, 'ai_tags': ai_labels, 'suggested_tags': suggested }
```

(工具/文件) : backend/utils/thumbs.py — 缩略图生成 (create_thumbnail)

```
def create_thumbnail(src_path, dst_path, max_size=256):
    img = Image.open(src_path)
    if img.is_animated: img = first_frame
    img = img.convert('RGB')
    thumb = img.resize(new_size, Image.LANCZOS)
    thumb.save(dst_path, format='JPEG', quality=80)
```

(路由/文件) : backend/routes/image_routes.py — 图片列表与 Hot 排序 (关键排序逻辑)

```
images.sort(key=lambda img: (
    (img.likes or 0) + (img.views or 0),
    img.likes or 0,
    img.upload_time
), reverse=True)
```

(路由/文件) : backend/routes/image_routes.py — 编辑 (edit_image : 替换文件 + 更新 tags)

```
def edit_image(id):
    if 'tags' in form: image.tags = new_tag_list
    if 'file' in files:
        remove(old_file); save(new_file); create_thumbnail(...)
    db.commit()
```

3.3 数据库设计

数据库采用 SQLite (文件存放在 `backend/data/`)。主要表结构 (概要) :

- `User` : `id, username, email, password_hash, avatar_path, create_time`。
- `Image` : `id, user_id, filename, thumbnail_filename, upload_time, exif_time, resolution_width/height, views, likes`。
- `Tag` 与 `ImageTag` : 标签表与图片-标签关系 (多对多)。

详细设计参见 [Design.md](#) 的第 5 节。

4 API 摘要 (常用)

- `POST /api/register` : 注册 (`username, email, password`)。
- `POST /api/login` : 登录, 返回 `access_token`。
- `POST /api/images` : 上传图片 (需要 JWT, 表单字段 `file`, 可选 `tags`)。实现见 [backend/routes/image_routes.py#L1](#)。
- `GET /api/images` : 列出图片, 支持 `sort, tag, username, image_id` 等查询参数。
- `POST /api/images/analyze` : 临时分析图片 (返回 EXIF、AI 标签、建议标签)。

5 部署与运行

开发运行:

后端 (Windows PowerShell 示例) :

```
cd backend
python -m venv venv
.\venv\Scripts\Activate.ps1
pip install -r requirements.txt
python app.py
```

前端:

```
cd frontend  
npm install  
npm run dev
```

使用 Docker Compose (推荐一键启动，全局在项目根目录运行)：

```
docker-compose up --build
```

更多配置与部署说明见 [README.md](#) 与 [docker-compose.yml](#)。

6 实验与测试

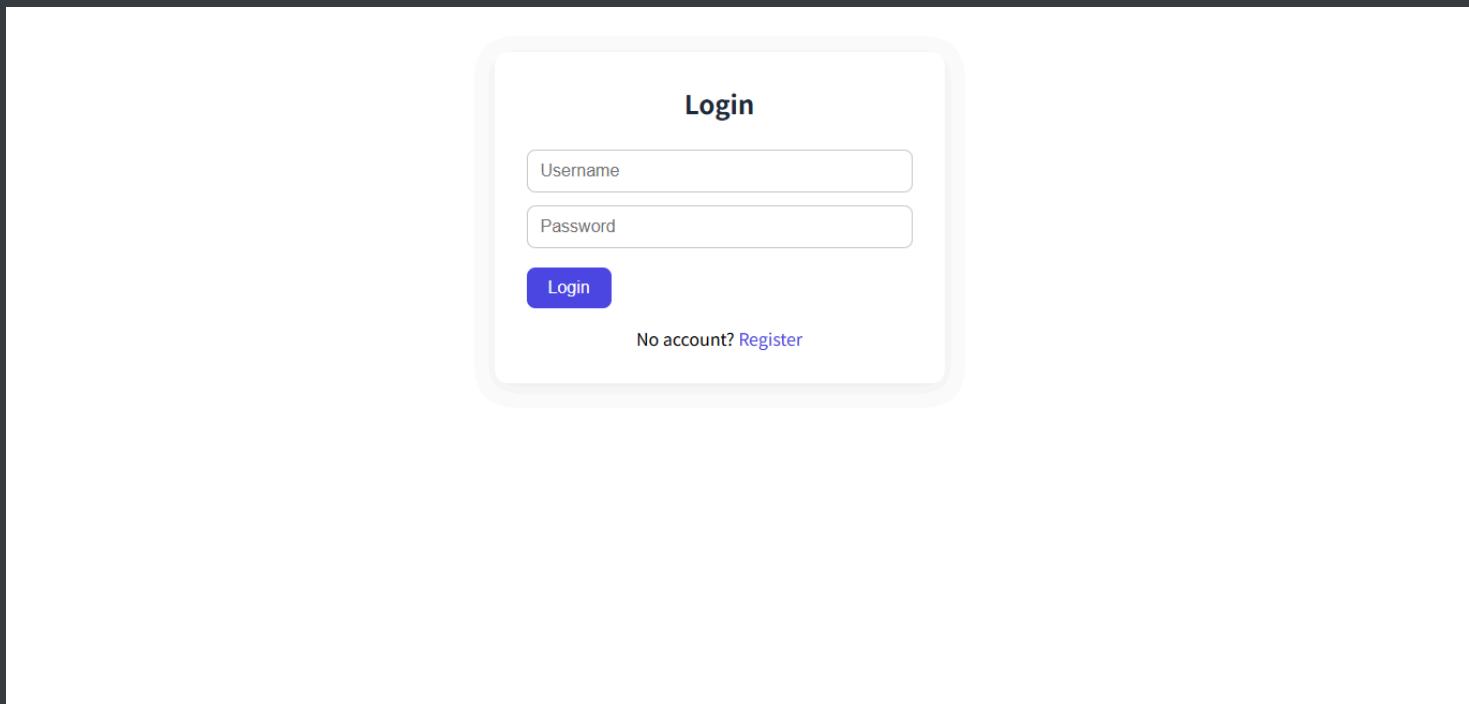
测试模块	功能点	测试结果	备注
用户管理	注册功能	<input checked="" type="checkbox"/> 通过	支持用户名/邮箱唯一性校验
用户管理	登录功能	<input checked="" type="checkbox"/> 通过	返回JWT令牌
图片处理	文件上传	<input checked="" type="checkbox"/> 通过	支持拖拽和点击上传
图片处理	格式校验	<input checked="" type="checkbox"/> 通过	支持JPG/PNG/GIF格式
图片处理	EXIF解析	<input checked="" type="checkbox"/> 通过	自动提取拍摄时间和GPS信息
图片处理	缩略图生成	<input checked="" type="checkbox"/> 通过	提升加载速度
数据管理	数据库记录	<input checked="" type="checkbox"/> 通过	完整记录图片信息
查询功能	多条件筛选	<input checked="" type="checkbox"/> 通过	支持标签、时间、用户、ID查询
图片操作	编辑功能	<input checked="" type="checkbox"/> 通过	支持旋转、翻转、滤镜、裁剪
图片操作	删除功能	<input checked="" type="checkbox"/> 通过	支持确认提示
图片操作	点赞功能	<input checked="" type="checkbox"/> 通过	热度算法计算

本地测试结论：

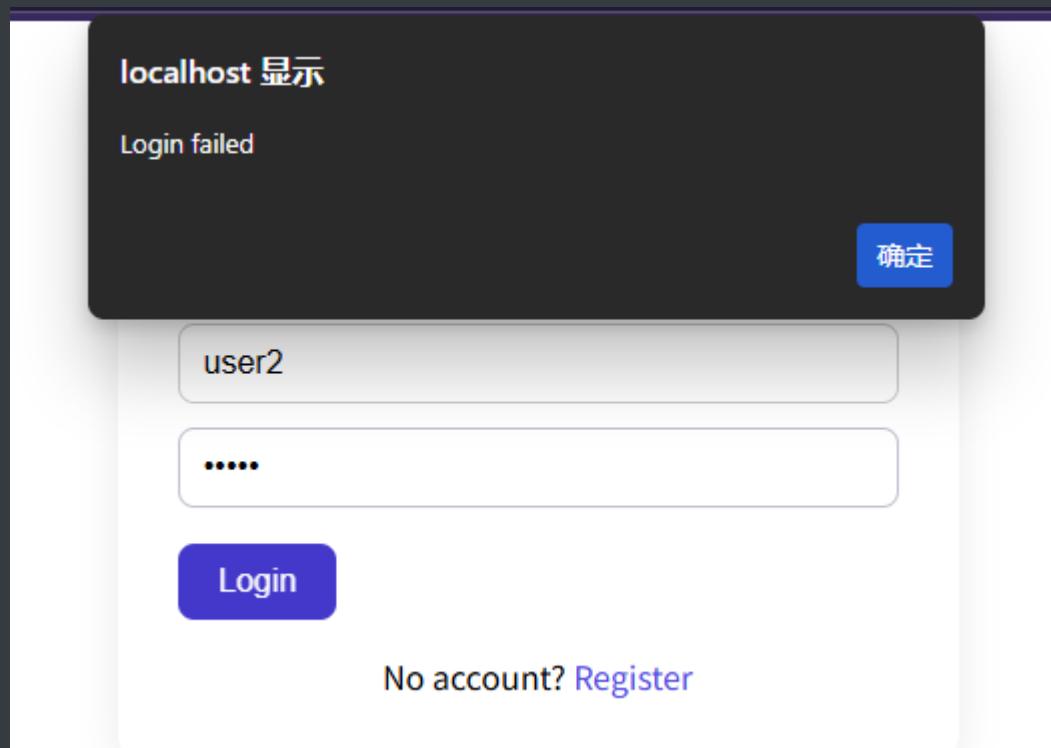
1. 前后端API通信正常，图片上传展示流程完整
2. EXIF信息解析准确，GPS信息可转换为标签建议
3. 缩略图机制显著提升图库加载性能
4. 所有核心功能模块运行稳定

我们接下来逐个模块讲解我们实现的效果：

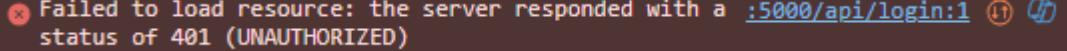
1. login



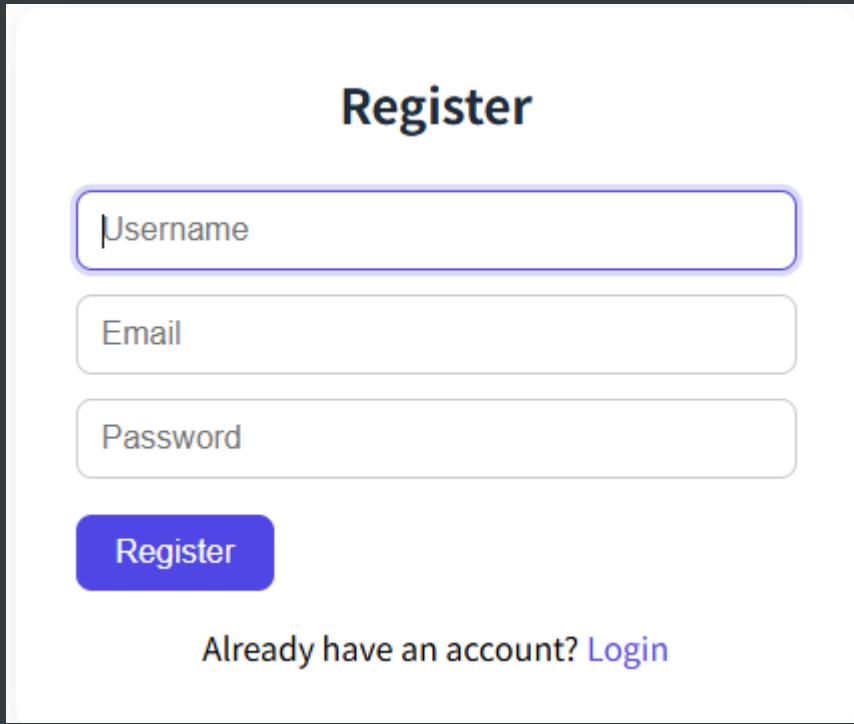
这是我们的login界面，需要输入username和password,点击Login登陆。点击 Register 即可跳转 Register



对于不存在的用户和错误的密码，均会alert：“Login Failed”

后端均会返回 

2.Register

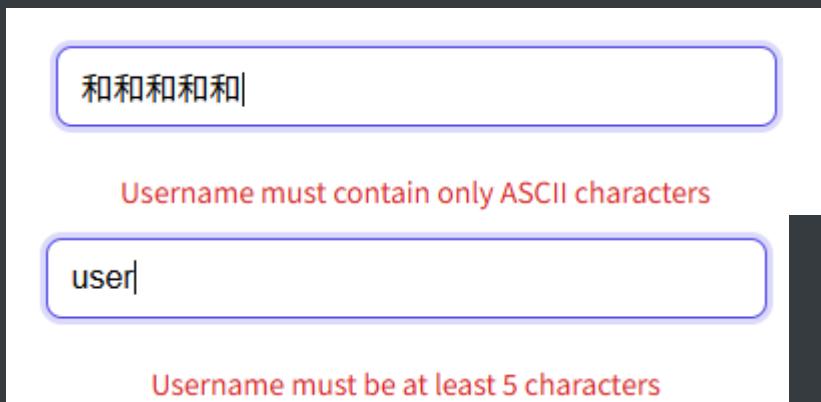


The image shows a registration form titled "Register". It contains three input fields: "Username", "Email", and "Password", each with a placeholder text. Below the inputs is a blue "Register" button. At the bottom, there is a link "Already have an account? [Login](#)".

这是Register界面，需要输入username, Email, Password来register，点击 Login 即可跳转登陆

Register在前端也设置了对应的格式检查：

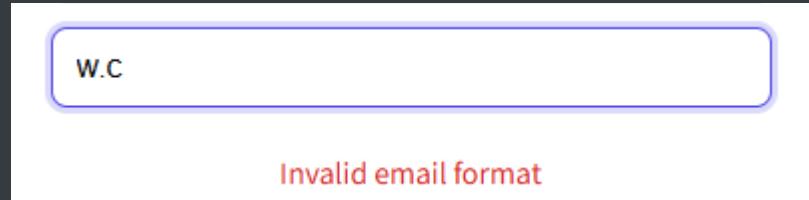
username要求必须长度不小于5个字符，且必须是ASCII码



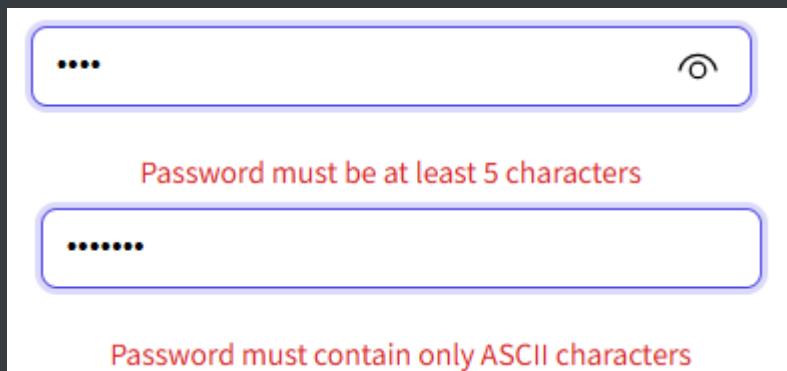
The image shows the registration form with validation errors. The "Username" field contains "和和和和和" and has a red error message "Username must contain only ASCII characters". The "Email" field contains "user" and has a red error message "Username must be at least 5 characters".

| 两种不合法的输入

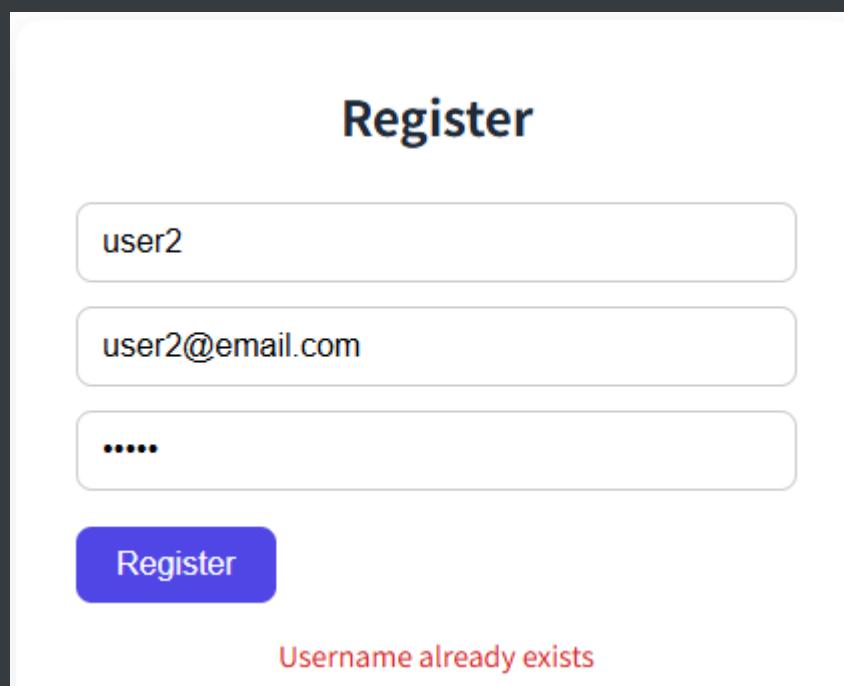
Email采用了简单的格式校验



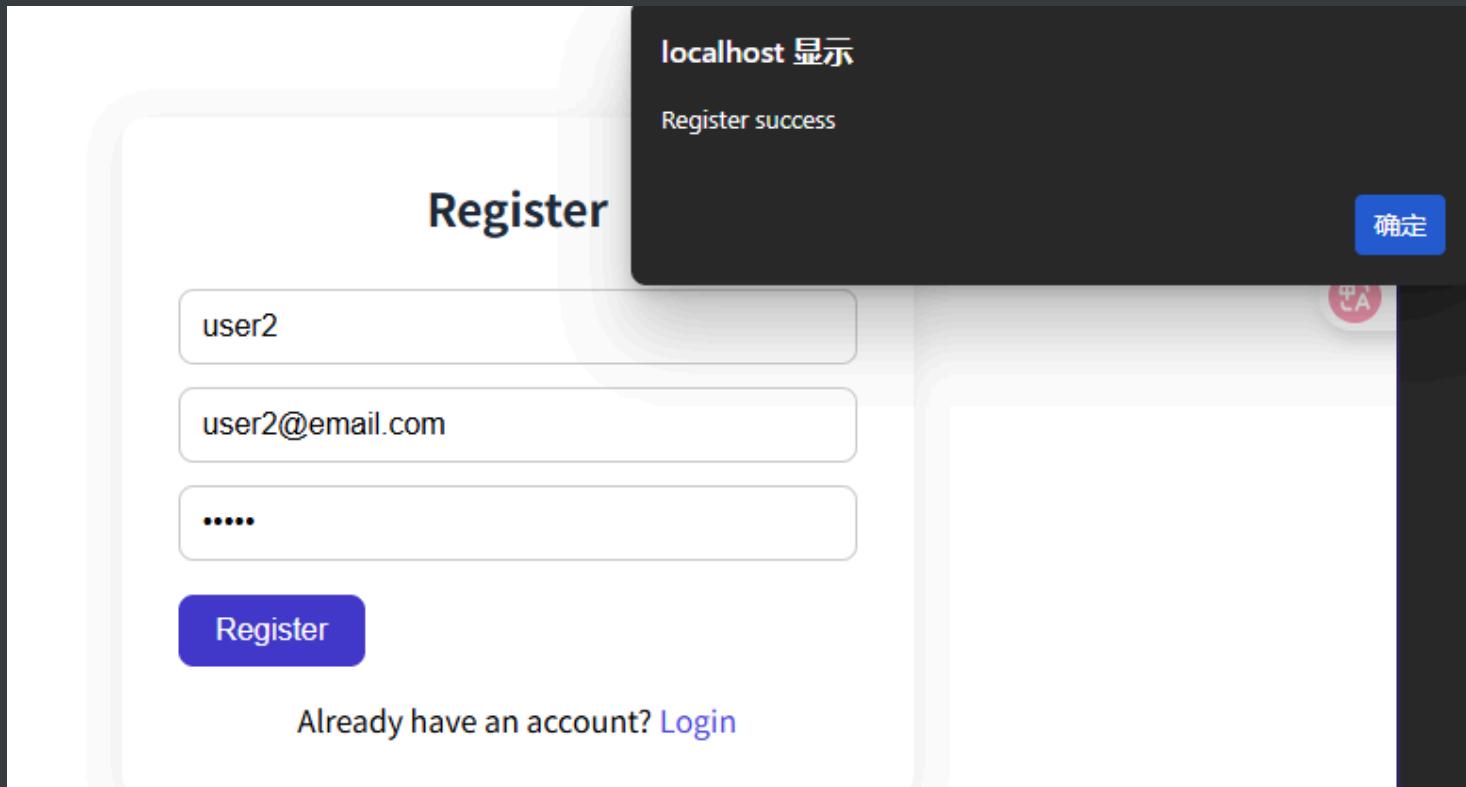
Password的要求与username相同，同时做了隐藏机制，贴近正式网站



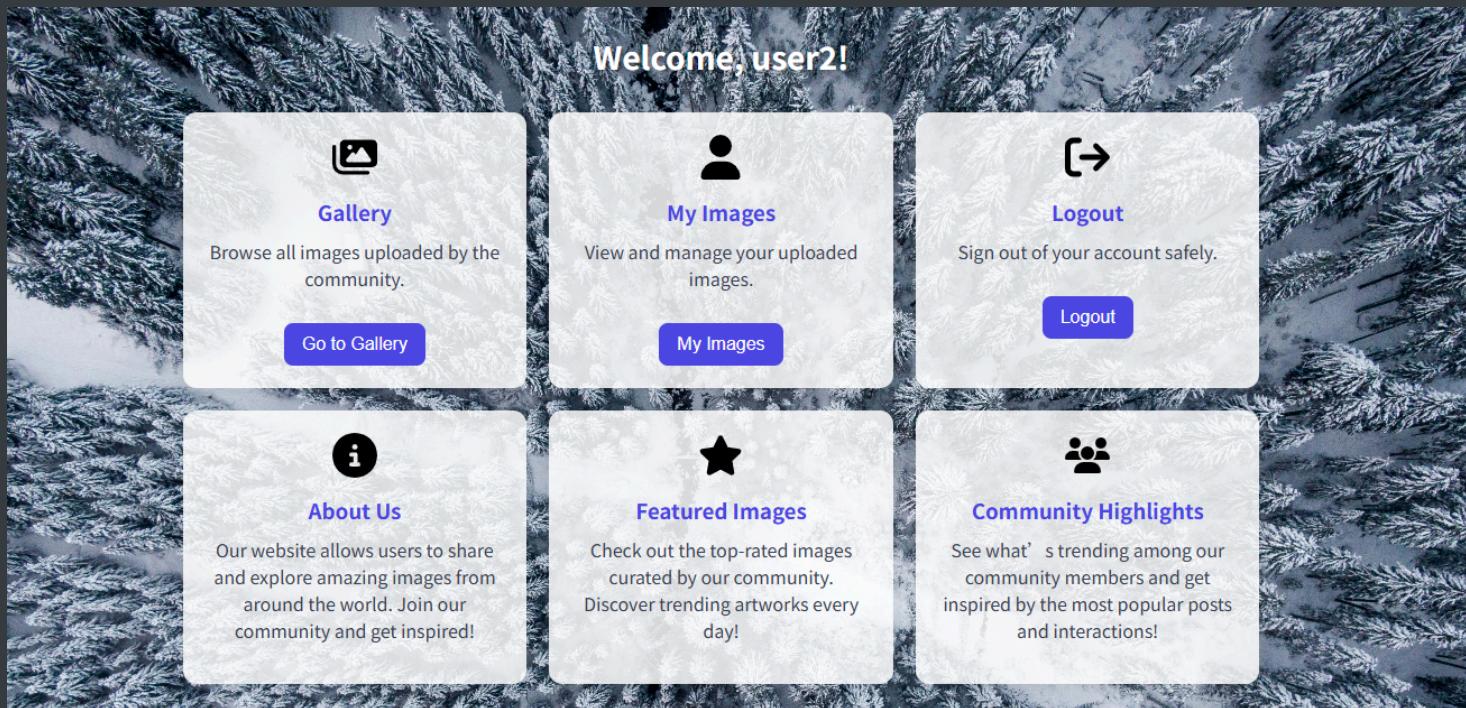
同时，基于用户名和邮箱唯一性的要求，如果register中使用的用户名或者邮箱是数据库中拥有的，则会显示返回注册失败



register成功后会alert:



3.Home Page



主页集中展示了关于我们网站的介绍，以及三个最基本的去向：Gallery（图库），My Image（用户自己上传的图片），Logout（退出登陆）

4. My Images



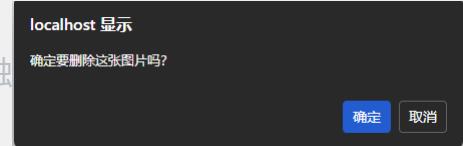
My Images是浏览用户所上传的所有的图片的地方，在未上传任何图片的时候，显示为“你还没有上传任何图片”。可以通过[上传新图片](#)

点击进入Upload模块



上传图片后会逐张显示，每张图片均有delete button和edit button，点击 编辑 即可进入Edit界面。此时
图片为缩略图，点击图片即可查看原图

点击 删除 即可删除此图片，删除时会有对应的确认提示，以防误触



每张图片在鼠标悬停时会自动模糊并告知显示该图片所拥有的标签。出于数量上的显示限制，每张图
片最多显示8个tag，为其最晚添加的8个tag



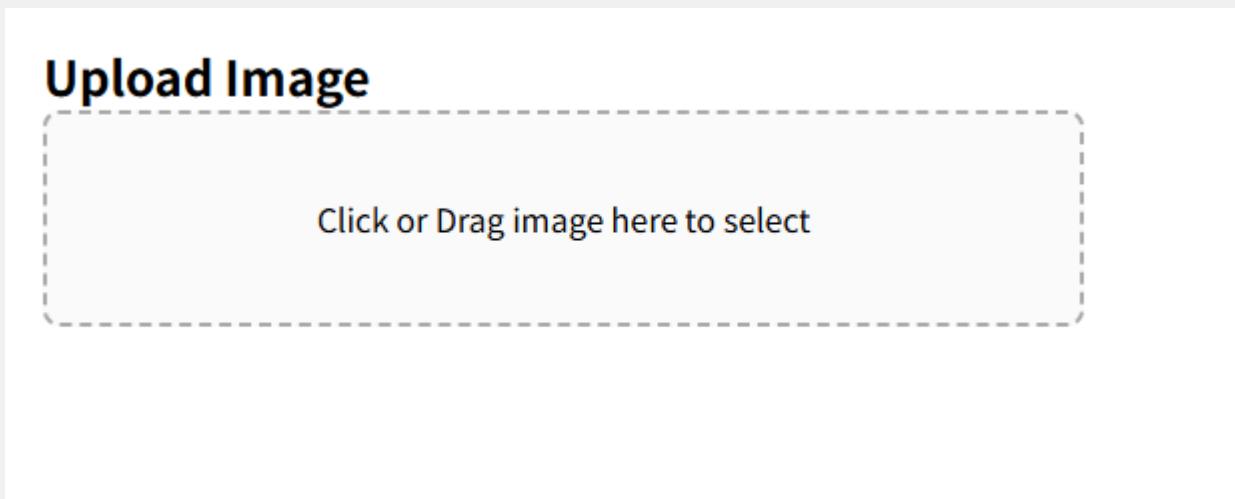
展示界面同样也是手机端友好的

我的图片



手机端的每行图片数量会根据屏幕宽度自行调整，图片显示的tag会在点击访问图片后长期显示

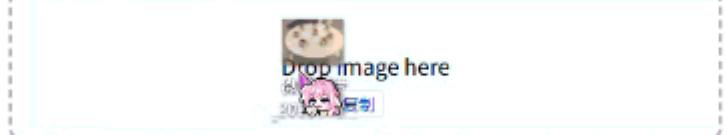
5.Upload Images



进入Upload Image模块，最开始展现的是单一的上传区域。

上传区域采用了PC端友好的设计，允许用户通过拖拽的方式将Image Upload。

Upload Image



当然,出于对手机端友好的考虑,同样允许通过点击上传区域,然后选择目录下文件的方式上传文件

Upload Image

Click or Drag image here to select



← → ⏪ ⏩

图片 >

上传后会出现预览图

Upload Image

Click or Drag image here to select



Analyze Tags

Add tag and press Enter

Popular Tags

#test

Confirm & Upload

图片上传支持jpg, png, gif格式, 对于不支持的格式, 会显示上传失败

Upload Image

Click or Drag image here to select



localhost 显示

Upload failed

确定

点击Analyze Tags会得到后端通过EXIF解析和AI解析图片得到的suggested tags,同时也会显示这张图片中包含的基础的EXIF信息



Analyze Tags

Resolution: 4608 × 3456
Time: 2025:12:20 19:47:24
Location: 30.2704° N, 120.1182° E

Suggested Tags

#4K #屏幕截图 #书房 #vivo_s17e #2025-12 #night
#location #酒窖 #vivo #has_gps #north_hemisphere
#device #户型图 #landscape #骨灰盒 #2025

可以通过点击suggested tags 来添加标签,也可以通过手动输入tag_name,回车输出tags,采用','作为手动输入tags的间隔符, 点击tag上的 ✕ 即可删除此tag

#night ✕ #4K ✕ #户型图 ✕ #ZJU ✕

Add tag and press Enter

同时,网站会统计本网站所有标签中,最热门的前10个tags,作为popular Tags提供给user选择。

Popular Tags

#test

本站所有图片的标签均有user自己填入, 不采用自动填入标签的方式, 提供user极大的自主性, 允许不设置tags

所有tags填写完毕后, 点击

Confirm & Upload

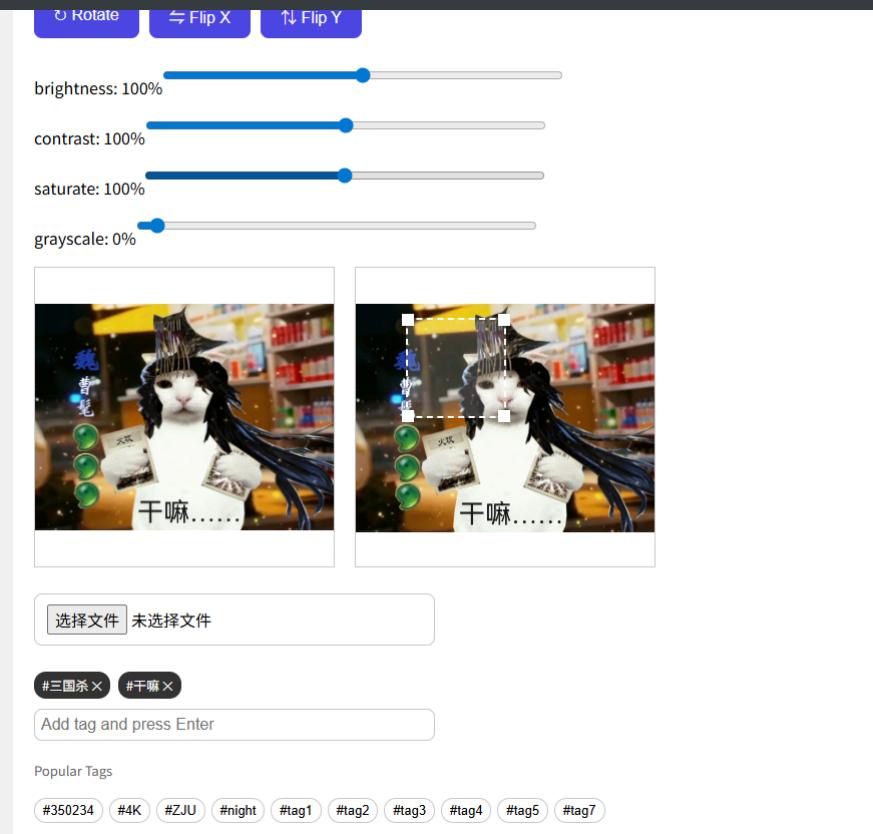
即可成功上传图片, 上传成功后会自动回到

My Image

6.Edit

编辑页面支持了基本的一些图像编辑, 并且支持图像编辑预览

Advanced Edit



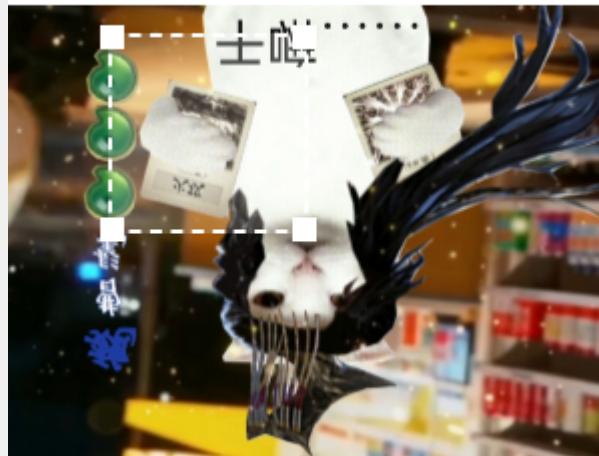
1 Rotation

点击 **© Rotate** 即可使图像顺时针旋转90°（即 **© Rotate** 标明的方向），预览效果如下



2 Flip

点击 **⏵ Flip X** **⏵ Flip Y** 分别可以实现水平方向和垂直方向的翻转，预览效果如下



3. Filter

网站提供了最基础了几种滤镜：亮度，对比度，饱和度，灰度。

brightness: 100%

contrast: 100%

saturate: 100%

grayscale: 0%

The image shows a control panel with four horizontal sliders. Each slider has a blue circular handle and a numerical value indicating its current setting. The first slider is labeled "brightness: 100%", the second "contrast: 100%", the third "saturate: 100%", and the fourth "grayscale: 0%".

采用滑动栏调节。由于此处的组合过多，预览效果只显示灰度变化：



4. cropper

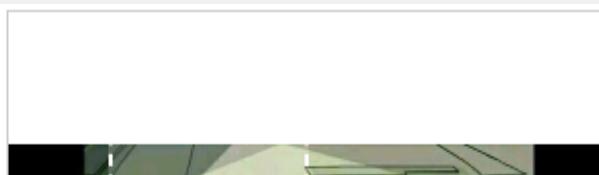
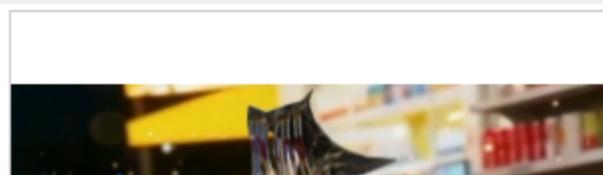
提供基础的自由裁剪，裁剪框由手动调整，若编辑过程中不想发生裁剪，即可将裁剪框拉至最大即可。裁剪设计中要求不得超出图片最大范围。

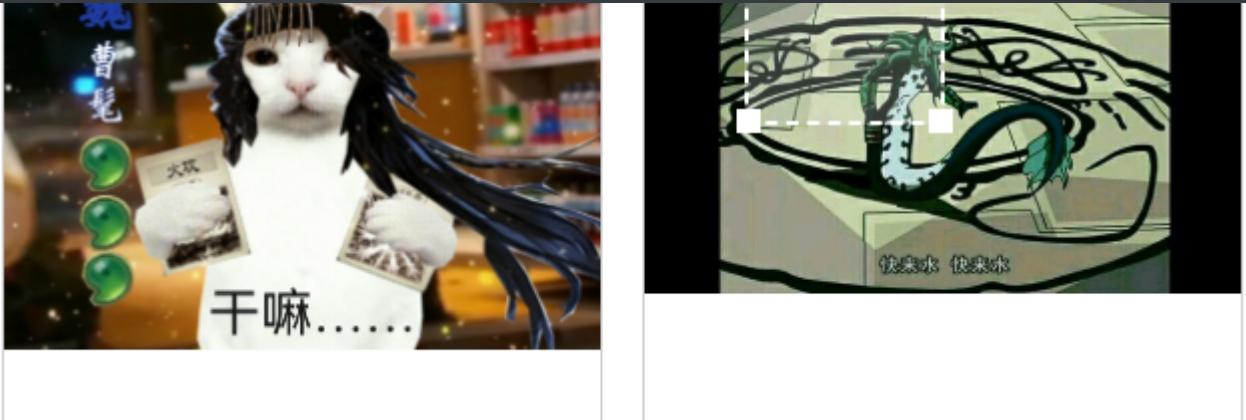


5.File Exchange

如果之前替换之前上传的文件，可以通过

直接选择新的文件，预览效果如下





6. Tags alternate

除了没有Analyze Tags外，Edit页面提供了和upload页面一样的改变tags的方式

#三国杀 X #干嘛 X

Add tag and press Enter

Popular Tags

#350234 #4K #ZJU #night #tag1 #tag2 #tag3 #tag4 #tag5 #tag7

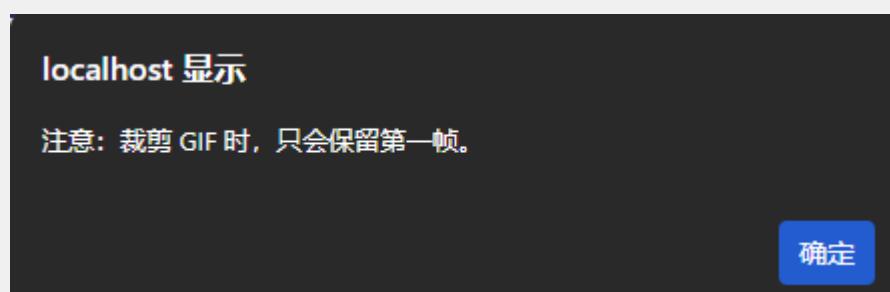
7. save or cancel

完成上述编辑后，点击 **Save Changes** 即可完成此次编辑，图片会采用替换的形式完成此次的编辑。

当然，如果你对此次编辑不满意，可通过点击 **Cancel** 放弃此次的所有编辑

8. 额外说明

上述Edit内容的支持格式为jpg, png等静态图片形式；对于动态图片，如gif，Edit后仅保留第一帧；Edit gif时，Edit界面会给出提示。



7.Gallery

通过Home界面的 [Go to Gallery](#) 将来到gallery界面

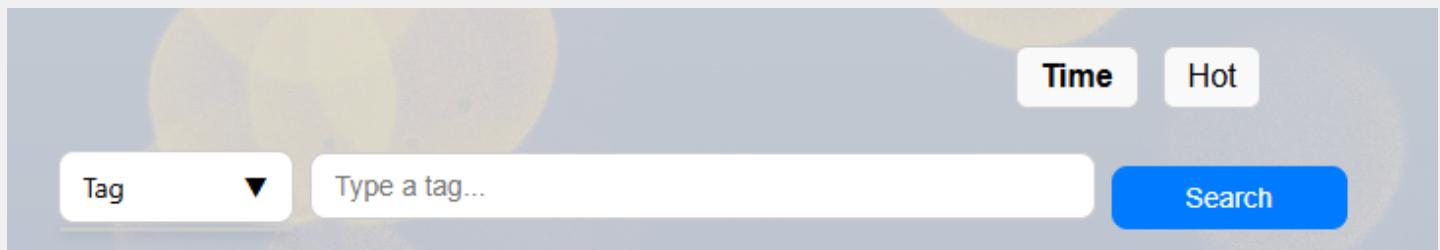
Gallery界面分为轮播层，搜索层和展览层

1. 轮播层



轮播层每三秒切换一次图片，现在主要用于展示热度最高的前五张图片，每3秒切换一张图片。电脑端可采用拖拽和点击索引点的形式更改图片，手机端可采用拖拽的方式切换图片

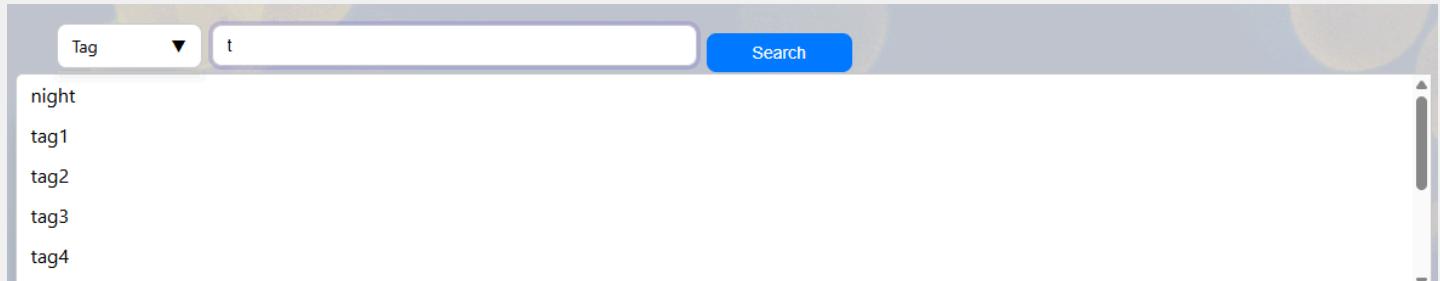
2. 搜索层



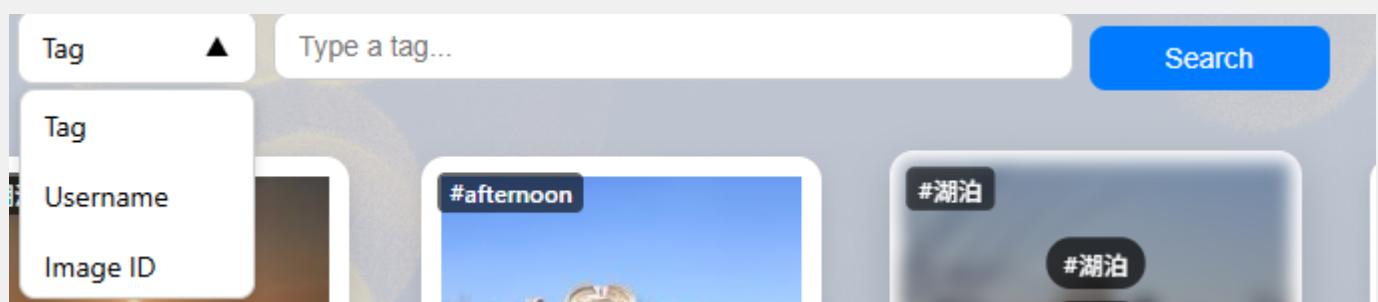
搜索层本质上是对展示图片的一种控制逻辑，现在主要由排序控制和显示控制两部分组成。

排序控制：点击 **Time** 会采用上传时间的降序排列，点击 **Hot** 会采用热度的降序排列，其中热度算法将在展示层提到

显示控制：提供了三种搜索的方式：通过tag搜索，通过上传的用户名来搜索，通过图片ID来搜索。其中tag搜索采用了模糊匹配和下拉式建议



搜索模式选择也采用了下拉式显示，旨在为用户提供更好的视觉体验



排序控制是在显示控制之后起到控制作用的，即页面图片的布局是按照先通过找到符合条件的图片，再按照选定的排序顺序展示的

3 展示层



展示层的图片与在My Image中的图片相比，增加了几个性质：

1. 在图片的左上角，会标定一个主标签，主标签为该图片使用的所有tags中全站使用次数最多的标签
2. 图片的左下角和右下角分别有 view 和 like 。点击图片后，view次数+1；非图片上传者点击❤后，图片like次数+1；热度算法f(p)基于图片的view,like,upload_time构成，满足
 1. 对于图片p1和p2,p1.view+p1.like>p2.view+p2.like $\Rightarrow f(p1) > f(p2)$;
 2. $f(p) = \frac{1}{1 + e^{-\alpha((p.view + p.like) - \beta * p.upload_time)}}$

- 2. $p1.view + p1.like = p2.view + p2.like, p1.like > p2.like \Rightarrow f(p1) > f(p2)$,
- 3. $p1.view + p1.like = p2.view + p2.like, p1.like = p2.like, p1.upload_time > p2.upload_time \Rightarrow f(p1) > f(p2)$

热度算法将用于搜索层的Hot排序和轮播层的图片选择

展示页面一样对手机端友好，采用了和My Image端相同的友好设置，这里不再赘述

7 遇到的问题与解决

- EXIF 解析失败：不同图片的 EXIF 字段不一致，使用 `exifread` 并对缺失字段安全降级（返回 `None`）。
- 跨域 (CORS)：开发时启用 `flask_cors.CORS(app)` 以便前后端分离调试。
- AI 接口限制：示例使用百度 API 需替换有效密钥；未配置时仅返回 EXIF 推荐标签。

8 总结与展望

该项目已实现图片上传、EXIF 解析、缩略图生成、标签管理与基本检索功能，完成了前后端分离的原型系统。未来可扩展方向包括：

- 引入更强的 AI 标签模型（如 CLIP 或自训练模型）提升语义检索能力；
- 将 SQLite 迁移为 MySQL 或 PostgreSQL 以适应更大并发场景；
- 使用云存储（如 S3）替代本地文件存储以提升可用性与扩展性；
- 为图片编辑引入前端实时预览与 WebAssembly 加速图像处理。

参考资料

- Design 文档：[Design.md](#)
- 项目说明：[README.md](#)
- 源码：`backend/` 与 `frontend/`

附录：关键文件位置

- 后端入口：[backend/app.py](#)
- 图片路由与实现：[backend/routes/image_routes.py](#)
- 前端入口：[frontend/src/main.jsx](#)

