Windows SD 卡照片整理与拍摄总结工具（MVP 版）

项目组织与开发说明书（对齐协作指南 v1.1：FastAPI + Vue 3（Vite）+ MySQL（Docker Compose））

版本：v2.1 | 日期：2026-01-12 | 平台：Windows 10/11（本地开发与演示）

0. 阅读指南（先看这一页）

- 本说明书面向“有编程基础但缺乏软件工程经验”的 4 人小组，目标是在 5 天内做出可演示、可复现的最小可行产品（MVP）。
- 本说明书的技术栈、目录结构、环境变量命名、启动命令与《VS Code + GitHub 协作指南（FastAPI + Vue + MySQL）》保持一致；如两份文档存在差异，以协作指南为准。
- MVP 只做本地导入整理与拍摄总结：不做发布/社交，不做自动插卡监听（用户手动选择 SD 目录）。
- AI 调用按需触发：只有点击“AI 分类/生成总结”时才调用多模态大模型 API；并发用 4 线程即可。
- 交付物：可运行的 GitHub 仓库（前后端 + Docker Compose）、演示视频/截图、分工与测试报告、本文档。

1. 项目概述与 MVP 范围

你们的需求可抽象为：从 SD 卡导入照片 → 读取 EXIF → 按“日期/类别”整理到本地图库 → 同步 JPG/RAW → 统计并生成拍摄总结（图表 + 文字）。

1.1 MVP 必做功能（5 天内必须完成）

- 手动选择 SD 卡目录与本地图库根目录（Library）。
- 扫描照片并提取 EXIF：拍摄时间、机身、镜头、焦段、ISO、光圈、快门（缺失字段允许为空）。
- 同名配对 JPG 与 RAW（例如 IMG_0001.JPG ↔ IMG_0001.ARW/DNG/CR2/NEF 等）。
- 按规则复制到 Library：Library/YYYY-MM-DD/类别/（默认类别“未分类”）。
- 将照片信息写入 MySQL（用于统计、查询与避免重复导入）。
- 前端展示缩略图墙、筛选（日期/类别/焦段/ISO），并支持勾选“精选”。
- 导出精选：同步导出 JPG + RAW（若存在），可选打包 ZIP。
- 生成拍摄总结：统计图表 + 调用 LLM 输出文字复盘（针对本次导入或选择的日期范围）。

1.2 MVP 明确不做（避免踩坑）

- 不做“插卡自动识别/自动弹窗导入”。
- 不做跨平台（仅 Windows）。
- 不做账号体系、发布、点赞评论等社交功能。
- 不做 RAW 解码与 AI 识别：AI 只识别 JPG 缩略图；RAW 只做文件级同步。
- 不做复杂队列（Celery/Redis）。MVP 采用 FastAPI 后台任务/线程池 + 前端轮询即可。

2. 技术栈与环境（以协作指南为准）

2.1 技术栈（统一口径）

- 后端：FastAPI（Python 3.10/3.11），提供扫描/导入/查询/导出/AI/总结等 REST API。
- 前端：Vue 3 + Vite（Node.js 18/20 LTS），提供照片墙、筛选、进度提示、图表展示。
- 数据库：MySQL 8.0（Docker Compose 一键启动），用于记录照片元数据与统计。
- AI：多模态大模型 API（图片分类/标签）+ 文本 LLM（总结文案）。
- 协作：GitHub 仓库 + VS Code Git 插件（不做保护/审核，但仍用分支开发）。

2.2 必装软件清单（每人都要装）

- Git for Windows
- VS Code（并登录 GitHub）
- Python 3.10/3.11（组内统一一个版本）
- Node.js 18/20 LTS（组内统一一个版本）
- Docker Desktop（用于 MySQL，建议开启 WSL2）

2.3 一键启动（最小可运行路径）

目标：任何成员在新电脑上 30 分钟内跑起来。按以下顺序启动：MySQL → 后端 → 前端。

步骤：（1）启动 MySQL（在仓库根目录）

```
docker compose up -d

docker ps
```

步骤：（2）启动后端 FastAPI（在 backend 目录）

```
cd backend

python -m venv .venv

.\.venv\Scripts\activate

pip install -r requirements.txt

copy .env.example .env

uvicorn app.main:app --reload --host 127.0.0.1 --port 8000
```

验收：浏览器打开 http://127.0.0.1:8000/docs 可见接口文档。

步骤：（3）启动前端 Vue 3（在 frontend 目录）

```
cd frontend

npm install
```

```
copy .env.example .env

npm run dev
```

验收：浏览器打开 Vite 输出地址（通常是 http://127.0.0.1:5173）。

3. 系统结构与数据流（FastAPI + Vue）

3.1 推荐目录结构（monorepo）

```
project-root/
  backend/                  # FastAPI 后端
    app/
      main.py               # 入口：创建 FastAPI、挂载路由、CORS、静态文件
      api/
        routes/             # API 路由文件（只做参数校验与调用 service）
        schemas/            # Pydantic 入参/出参模型
      services/             # 业务逻辑（扫描/整理/AI/总结/导出）
      db/                   # 数据库连接、ORM 模型、CRUD
      core/                 # 配置、日志、通用工具（如路径、hash、缩略图）
    requirements.txt
    .env.example
  frontend/                 # Vue 3 + Vite 前端
    src/
      api/                  # axios 封装与接口调用
      pages/                # 页面（或 views/，组内统一）
      components/           # 可复用组件（照片卡片、筛选条、进度条）
    .env.example
    package.json
  docker-compose.yml        # MySQL 一键启动
  docs/                     # 文档与答辩材料
```

```
.gitignore

README.md
```

3.2 后端模块划分（建议按文件分工）

- backend/app/api/routes：API 路由（薄层），例如 photos.py、ai.py、summary.py、export.py。
- backend/app/services：业务逻辑（厚层），例如 scanner_service.py、organizer_service.py、ai_service.py、summary_service.py。
- backend/app/db：数据库（连接、模型、CRUD），例如 session.py、models.py、photos_repo.py。
- backend/app/core：通用工具（路径处理、hash、缩略图生成、配置读取、日志）。

3.3 前端模块划分（Vue）

- frontend/src/api：统一封装 axios（baseURL 来自 .env），所有接口调用集中管理。
- frontend/src/pages：页面（导入/照片墙/总结），页面只负责展示与交互。
- frontend/src/components：可复用组件（照片卡片、筛选器、分页/虚拟滚动、进度显示）。

3.4 主流程（从 SD 到总结）

1. 用户在前端选择 SD 目录与 Library 目录（通过后端文件选择能力：最简可先输入路径文本框）。
2. 前端调用后端“扫描”接口：后端遍历 SD 目录，找到 JPG，并同名匹配 RAW，读取 EXIF，生成缩略图（缓存）。
3. 后端将扫描结果写入 MySQL（以 sha1 去重），返回本次扫描统计与照片列表。
4. 用户点击“整理到 Library”：后端按 YYYY-MM-DD/类别 规则复制 JPG + RAW 到本地图库，并更新数据库 library_path。
5. 用户在照片墙勾选“精选”：前端调用更新接口写回 is_selected=1。
6. 用户点击“导出精选”：后端根据 is_selected 导出 JPG + RAW（若存在），可选打包 ZIP。
7. 用户点击“AI 分类/生成总结”：后端对缩略图批量调用多模态 API（4 线程），写回 category/tags；然后计算统计并调用 LLM 生成总结文案。

3.5 最小 API 列表（建议先做这些）

- GET /health：健康检查（前端联调用）。
- POST /photos/scan：扫描 SD 目录（入参：sd_path、options）。
- POST /photos/import：整理到 Library（入参：library_root、date_range 可选）。
- GET /photos：列表查询（支持 filters：date、category、is_selected）。
- PATCH /photos/{id}：更新单张（category/tags/is_selected 等）。
- POST /export/selected：导出精选（zip 可选）。
- POST /ai/classify：对一批照片做 AI 分类（可做任务化）。
- POST /summary/generate：生成统计图表数据与总结文案。

4. 前端界面与功能设计（Vue 3）

4.1 页面结构（建议 3 个页面即可）

- 导入页：输入/选择 SD 目录与 Library 目录；按钮：扫描、整理。
- 照片墙页：缩略图瀑布流/网格；筛选（日期/类别/焦段/ISO）；勾选精选；导出精选。
- 总结页：统计图表（数量、焦段、ISO、快门、类别占比）+ LLM 文字复盘。

4.2 用户操作流程（演示脚本）

8. 打开前端页面 → 输入 SD 目录与 Library 目录 → 点击【扫描】。
9. 看到缩略图墙与统计：总数、RAW 匹配数、重复数。
10. 点击【整理到 Library】→ 硬盘生成 YYYY-MM-DD/未分类/ 并复制文件。
11. 勾选若干张 → 点击【导出精选（含 RAW）】→ 生成导出目录或 ZIP。
12. 点击【AI 分类】→ 类别/标签更新；点击【生成总结】→ 展示图表 + 文字复盘。

5. 数据库设计（MySQL：最小字段集）

目标：记录导入过的照片、EXIF 字段、RAW 路径、AI 分类/标签与精选标记，用于统计与避免重复导入。MVP 建议先只做一张 photos 表。

5.1 photos 表（DDL，直接可用）

```
CREATE TABLE IF NOT EXISTS photos (
  id            BIGINT PRIMARY KEY AUTO_INCREMENT,
  file_name     VARCHAR(255) NOT NULL,
  file_path     TEXT NOT NULL,              -- 原始 JPG 路径（SD 卡中）
  raw_path      TEXT NULL,                  -- 若匹配到 RAW，则写入
  library_path  TEXT NULL,                  -- 整理后在 Library 的 JPG 路径
  taken_at      DATETIME NULL,              -- EXIF 解析失败可为空（可降级用文件时间）
  camera_model  VARCHAR(255) NULL,
  lens          VARCHAR(255) NULL,
  focal_length  FLOAT NULL,
  iso           INT NULL,
  aperture      FLOAT NULL,
  shutter       FLOAT NULL,                 -- 秒，例如 0.008=1/125
  category      VARCHAR(50) NOT NULL DEFAULT '未分类',
  tags_json     JSON NULL,                  -- 例如 ["海边","日落"]
```

```
  is_selected   TINYINT NOT NULL DEFAULT 0,

  sha1          CHAR(40) NOT NULL,          -- JPG 内容 hash，用于去重

  created_at    DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,

  updated_at    DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,

  UNIQUE KEY uk_photos_sha1 (sha1),

  KEY idx_photos_taken_at (taken_at),

  KEY idx_photos_category (category),

  KEY idx_photos_selected (is_selected)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

5.2 最小 DB 操作（必须实现的 4 类）

- init_db / 迁移：MVP 可在启动时执行 DDL（或用 Alembic，非必需）。
- upsert_photo：按 sha1 去重插入（重复则跳过或更新 file_path/raw_path）。
- list_photos：按日期范围/类别/精选状态查询，用于照片墙展示。
- update_photo：更新 category/tags_json/is_selected/library_path 等字段。

6. 关键实现细节（后端为主）

6.1 扫描与 RAW 配对（只以 JPG 为主）

- 原则：以 JPG 为“主文件”。找到 JPG 后，再在同目录按同名匹配 RAW。
- 不要遍历所有 RAW 去做识别：RAW 只做同步拷贝。

```
RAW_EXTS = [".arw", ".cr2", ".nef", ".dng", ".raf", ".rw2"]


for jpg_path in find_all_jpg(sd_root):
    stem = jpg_path.stem  # IMG_0001
    raw_path = None
    for ext in RAW_EXTS:
        candidate = jpg_path.with_suffix(ext)
        if candidate.exists():
            raw_path = candidate
            break
```

```
    exif = read_exif(jpg_path)

    sha1 = sha1_file(jpg_path)  # 分块读取

    upsert_photo_to_mysql(...)
```

6.2 文件整理：先复制（安全）

- MVP 建议只做复制（copy2），避免误删 SD 卡原片；稳定后再加“移动/删除源文件”模式。
- 目录规则：<Library 根目录>/YYYY-MM-DD/类别/。

```
dest_dir = library_root / date_str / category

dest_dir.mkdir(parents=True, exist_ok=True)


copy2(jpg_path, dest_dir / jpg_path.name)

if raw_path:

    copy2(raw_path, dest_dir / raw_path.name)
```

6.3 缩略图生成与访问

- 每张 JPG 生成一个 512px 宽缩略图（保持比例），保存到后端可控目录（例如 backend/storage/thumbs/）。
- 缩略图文件名建议用 sha1.jpg，避免同名冲突。
- 后端用 StaticFiles 挂载缩略图目录，前端直接用 URL 展示（避免把图片二进制塞进 JSON）。

6.4 AI 并发调用（4 线程 + 重试）

- 不引入 Celery/Redis。用 ThreadPoolExecutor(max_workers=4) 并发请求多模态 API。
- 每张图片失败可重试 2 次；最终失败写入错误字段（可选）并继续处理其他照片。
- 默认不自动调用 AI；只有用户点击按钮才批处理。

6.5 前后端联调要点（新手必看）

- 前端 baseURL 从 frontend/.env 读取（VITE_API_BASE_URL），不要写死。
- 后端开启 CORS（开发期允许 http://127.0.0.1:5173）。
- API 返回统一格式：data + message + error（可选），便于前端处理。

7. 多模态大模型 API 接入（最小实现）

7.1 统一返回格式（建议存入 tags_json）

```
{

  "category": "风光",
```

```
  "tags": ["海边", "日落", "逆光"],
  "caption": "海边日落的剪影画面",
  "confidence": 0.78
}
```

7.2 类别体系（先固定，避免越做越散）

- 人像、风光、街拍、建筑、美食、夜景、动物、活动、微距、未分类（共 10 类）

7.3 图片分类 Prompt 模板（可直接复制）

你是一名摄影内容标注助手。请根据输入图片完成如下任务，并只输出 JSON：

1) 从固定类别列表中选择最匹配的 category：

[人像，风光，街拍，建筑，美食，夜景，动物，活动，微距，未分类]

2) 给出 3-6 个中文 tags（短词/短语）

3) 给出一句 caption（不超过 25 字）

4) 给出 confidence（0-1）

输出 JSON 示例：

```
{"category":"风光","tags":["海边","日落","剪影"],"caption":"海边日落剪影","confidence":0.78}
```

7.4 总结文案 Prompt 模板（喂统计数据，不喂原图）

你是一名摄影教练。请基于以下统计数据生成“拍摄复盘”，输出为中文，结构如下：

- 今日主题概览（2-3 句）
- 数据解读（要点：焦段、ISO、快门、光圈的主要分布与结论）
- 3 条可执行建议
- 精选推荐（1 句）

统计数据（JSON）：

```
{{stats_json}}
```

8. 配置文件与启动说明（与协作指南一致）

8.1 后端 backend/.env.example（示例字段）

```
APP_HOST=127.0.0.1

APP_PORT=8000


MYSQL_HOST=127.0.0.1

MYSQL_PORT=3306

MYSQL_USER=photoapp

MYSQL_PASSWORD=photoapp_pwd

MYSQL_DB=photoapp


AI_API_KEY=PLEASE_FILL_ME

AI_BASE_URL=PLEASE_FILL_ME

AI_MODEL=PLEASE_FILL_ME
```

8.2 前端 frontend/.env.example（Vite）

```
VITE_API_BASE_URL=http://127.0.0.1:8000
```

8.3 docker-compose.yml（MySQL 一键启动）

```
version: "3.8"

services:

  mysql:

    image: mysql:8.0

    container_name: photoapp-mysql

    restart: unless-stopped

    environment:

      MYSQL_ROOT_PASSWORD: root_pwd
```

```
      MYSQL_DATABASE: photoapp
      MYSQL_USER: photoapp
      MYSQL_PASSWORD: photoapp_pwd
    ports:
      - "3306:3306"
    command: ["--character-set-server=utf8mb4", "--collation-
server=utf8mb4_unicode_ci"]
```

9. 五天排期与分工（按此执行能收敛）

9.1 四人建议分工（尽量减少冲突）

- 同学 A（后端-扫描）：scanner_service（文件遍历、RAW/JPG 配对、EXIF 口径）。
- 同学 B（后端-DB/API）：MySQL 表、SQLAlchemy/CRUD、photos 列表与更新接口。
- 同学 C（前端-Vue）：导入页/照片墙/总结页，筛选与勾选逻辑，图表展示。
- 同学 D（后端-AI/总结）：多模态 API 封装、线程池并发、总结生成接口。

9.2 日计划（每日验收点）

- Day 1：后端扫描+EXIF+RAW 配对跑通（命令行/接口都行）。验收：/photos/scan 能返回照片列表。
- Day 2：MySQL 入库 + 文件复制整理。验收：/photos/import 后硬盘生成日期目录并复制文件，/photos 列表可查。
- Day 3：Vue 照片墙（缩略图展示 + 筛选 + 勾选精选）。验收：页面可用、能更新 is_selected。
- Day 4：AI 分类/标签 + 总结生成。验收：点击按钮后 category/tags 更新，生成图表与文案。
- Day 5：导出 ZIP、异常处理、文档与演示脚本。验收：完整流程演示一遍不崩溃。

10. 测试清单与验收标准（小学期常用）

10.1 功能测试（必测）

- 选择包含 200+ 照片的目录：扫描完成且后端不崩溃；前端可加载照片墙。
- 同名 RAW 匹配：随机抽 20 组，匹配正确。
- 重复扫描/导入同一目录：sha1 去重生效，重复数统计正确。
- 整理到 Library 后：目录结构正确；文件数量与源目录一致（允许过滤非图片）。
- 勾选精选并导出：导出目录包含 JPG 与对应 RAW（若存在）；ZIP 可正常解压。
- 生成总结：图表不报错；LLM 文案能返回可读中文。

10.2 异常与边界（至少测 5 条）

- SD 路径选错或被拔出：后端提示错误并返回可读 message，不崩溃。

- EXIF 缺失：taken_at 为空也能入库与整理（用文件时间降级）。
- 文件名含中文/空格：整理与导出不出错。
- AI API Key 不存在：接口返回明确错误，前端提示并跳过 AI 流程。
- AI 请求超时：重试后仍失败则记录失败并继续处理其它照片。

11. 新手避坑贴士（强烈建议全组阅读）

- 路径处理后端统一用 pathlib.Path，避免 Windows 反斜杠转义问题。
- 不要对 RAW 做 AI 识别：永远用 JPG 缩略图识别，再关联同名 RAW。
- API Key 放 .env 文件，仓库提交 .env.example；.gitignore 忽略 .env。
- 整理动作先做复制（copy），确保 SD 卡原片安全。
- 多人协作：尽量不要多人同时改后端 main.py / 前端全局配置文件；需要改在群里说一声。

12. 交给编程 AI 的落地指令（按文件生成，逐个验证）

12.1 后端：生成 app/main.py（入口与路由挂载）

将下面内容复制给编程 AI，用于生成后端入口文件：

```
请用 Python 3.11 + FastAPI 编写 backend/app/main.py，要求：

- 创建 FastAPI 应用并启用 CORS（允许 http://127.0.0.1:5173）

- 挂载路由：/health、/photos、/ai、/summary、/export（路由文件放在 app/api/routes/）

- 挂载静态目录：/static/thumbs -> backend/storage/thumbs（用于缩略图访问）

- 从 .env 读取配置（APP_HOST/APP_PORT/MYSQL_* / AI_*）

- 提供 GET /health 返回 {"status":"ok"}
```

12.2 后端：生成数据库连接与 photos 表 ORM/CRUD

```
请用 SQLAlchemy 编写：

1) backend/app/db/session.py：根据 MYSQL_* 环境变量创建 engine 与 SessionLocal

2) backend/app/db/models.py：定义 Photo ORM（字段参考文档第 5 章 photos 表）

3) backend/app/db/photos_repo.py：实现 upsert_by_sha1、list_photos(filters)、
update_photo(id, fields)

要求：

- 连接字符串使用 mysql+pymysql://

- 所有查询支持分页（page/page_size）
```

- tags_json 使用 JSON 类型（SQLAlchemy JSON）

12.3 后端：生成扫描/整理/导出服务与 API

请编写后端以下文件：

- app/services/scanner_service.py：扫描目录、RAW 同名匹配、EXIF 解析、sha1 分块计算、生成缩略图

- app/services/organizer_service.py：按 YYYY-MM-DD/类别 复制 JPG+RAW 到 Library 并更新 DB

- app/services/export_service.py：导出 is_selected=1 的照片（含 RAW），可选 zip

- app/api/routes/photos.py：POST /photos/scan, POST /photos/import, GET /photos, PATCH /photos/{id}

- app/api/routes/export.py：POST /export/selected

要求：

- 扫描以 JPG 为主，RAW 仅同名匹配

- EXIF 失败可降级为文件时间

- 复制优先（copy2），不要删除源文件

- API 返回统一格式：{"data":..., "message":"...", "error":null}

12.4 前端：生成 Vue 3 + Vite 页面与 API 封装

请用 Vue 3 + Vite 生成前端骨架（frontend/），要求：

- src/api/http.ts：封装 axios，baseURL 来自 import.meta.env.VITE_API_BASE_URL

- src/pages/ImportPage.vue：输入 SD 路径、Library 路径，按钮：扫描、整理

- src/pages/GalleryPage.vue：缩略图网格，筛选（日期/类别/焦段/ISO），勾选精选（调用 PATCH /photos/{id}）

- src/pages/SummaryPage.vue：调用 /summary/generate 展示图表数据与总结文案

- 图表可先用 ECharts（或最简用表格），保证 5 天能交付

- 所有接口调用集中在 src/api/ 下，页面不要直接写 axios