VS Code + GitHub 协作指南

FastAPI（后端）+ Vue 3（Vite 前端）+ MySQL（Docker Compose）

最终定稿版 v1.1（面向小白，5 天小组冲刺适用）

适用场景	4 人小组 / 5 天工期 / 本地开发与演示 / 暂不做发布
技术栈	后端 FastAPI（Python） + 前端 Vue 3（Vite） + 数据库 MySQL（Docker Compose）
协作方式	GitHub 仓库 + VS Code Git 插件；不做保护/审核，但仍采用“分支开发 + 定期合并 main”
目标	任何成员在新电脑上 30 分钟内跑起来；main 分支尽量保持可运行
安全约定	API Key 不进 Git；照片/导出 zip/数据库文件不提交；仅提交代码与小型配置

阅读方式建议：按顺序完成第 1-6 章即可开始协作开发；遇到问题查第 7-9 章。

1. 你们要做的事情（一句话）

把代码托管到 GitHub，每个人用 VS Code 拉取代码，在自己的分支开发，定期同步并合并到 main，从而实现多人协作开发。

2. 最重要的 3 条协作规则（务必遵守）

- 不要多人同时改同一个文件：尤其是后端入口、路由聚合文件、前端全局配置。需要改时先在群里说一声。

- 小步提交、小步合并：功能做到 30%-50% 就提交并 push，不要憋到最后一次性提交。
- 合并前先同步 main：开始一天工作前，以及准备合并前，都先 pull main，并把 main 合并到自己的分支。

3. 组长一次性初始化（只做一次）

组长负责创建仓库、搭好目录、提交示例配置与启动说明。其他成员从第 4 章开始。

3.1 创建 GitHub 仓库

1. 在 GitHub 创建新仓库（建议勾选 Add a README）。
2. 在 Settings -> Collaborators 邀请队友加入仓库。
3. 初始化目录结构（见 3.2）与基础文件（见 3.3-3.4），完成首次提交。

3.2 推荐目录结构（monorepo：一个仓库放前后端）

目录结构示例（建议照抄）

```
project-root/
  backend/                  # FastAPI 后端
    app/
      main.py               # 后端入口
      api/                  # 路由（接口）
      services/             # 业务逻辑（扫描/AI/统计）
      db/                   # 数据库连接与模型
    requirements.txt
    .env.example
  frontend/                 # Vue 3 + Vite 前端
    src/
    package.json
    .env.example
  docker-compose.yml        # MySQL 一键启动
  docs/                     # 文档与答辩材料
  scripts/                  # 一键启动脚本（可选）
  .gitignore
  README.md
```

3.3 必备文件：.gitignore 与 .env.example

原则：私密信息与大文件不要进 Git。每个成员本地创建 .env，仓库只提交 .env.example。

.gitignore 最小建议（可按需增删）

```
.env
backend/.venv/
backend/__pycache__/
frontend/node_modules/
*.db
data/
exports/
thumb_cache/
```

后端示例：backend/.env.example

```
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
```

前端示例：frontend/.env.example（Vite 使用 VITE_ 前缀）

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

3.4 MySQL：用 Docker Compose 一键启动（强烈推荐）

这样全组 MySQL 版本与账号完全一致，避免安装、权限、服务启动等坑。

docker-compose.yml 示例（仓库根目录）

```
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"]
```

提示：Windows 需要安装 Docker Desktop 并开启 WSL2 支持。若学校电脑无法安装 Docker，可退回本机安装 MySQL（全组统一版本与账号）。

4. 每个人第一次上手：安装与登录（10-20 分钟）

4.1 必装软件清单

- Git for Windows
- VS Code
- Python 3.10/3.11（组内统一）
- Node.js 18/20 LTS（组内统一）
- Docker Desktop（用于 MySQL）

4.2 在 VS Code 登录 GitHub

VS Code 左下角头像/Accounts -> Sign in to GitHub，完成浏览器授权。之后 push 时一般不会再频繁输密码。

5. 拉取代码（clone）与分支创建（每人必做）

5.1 用 VS Code 克隆仓库

4. 按 Ctrl+Shift+P 打开命令面板，输入 Git: Clone。
5. 粘贴 GitHub 仓库 HTTPS 地址。
6. 选择本地目录并打开项目。

5.2 建议的分支使用方式（不做审核也要分支）

不做保护/审核也建议用分支：每人每个功能一个分支，完成后合并到 main。这样冲突更少、回滚更方便。

创建分支（建议照抄）

```
# 开工前：更新 main
git checkout main
git pull

# 创建自己的功能分支
git checkout -b feat/scan-exif
```

6. 本地运行：三件事（MySQL、后端、前端）

目标：任何成员在新电脑上，按下面步骤能把项目完整跑起来。

6.1 启动 MySQL（Docker）

```
# 在仓库根目录
docker compose up -d

# 查看是否启动成功
docker ps
```

6.2 启动后端 FastAPI

在 backend 目录创建虚拟环境，安装依赖，复制 .env.example 为 .env。

```
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 能看到接口文档即成功。

6.3 启动前端 Vue 3（Vite）

```
cd frontend
npm install
copy .env.example .env
npm run dev
```

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

7. 日常协作：每天照做的标准流程（最重要）

7.1 开工前（1 分钟）

7. 切到 main 并拉取最新：git checkout main && git pull
8. 切回自己的分支：git checkout feat/xxx
9. 如果 main 有更新，把 main 合并到自己的分支（见 7.3）。

7.2 提交与推送（commit / push）

小步提交：每完成一个小功能（能跑）就提交一次。

```
git add .
git commit -m "feat: implement exif parser"
git push
```

7.3 同步 main（减少冲突，强烈建议每天至少一次）

在自己的分支上把 main 的更新合并进来（新手推荐 merge，不用 rebase）。

```
git checkout feat/xxx
git fetch origin
git merge origin/main
# 解决冲突（如有）后：
git push
```

7.4 合并到 main（两种方式）

方式 A：用 PR（推荐）。不需要审核，但 PR 能让你在合并前看到差异，减少误操作。

方式 B：本地直接 merge 到 main 再 push（不推荐但可用）。

方式 A（推荐）：PR 自己合并

10. 把分支 push 到 GitHub：git push
11. 在 GitHub 网页点 Compare & pull request，创建 PR。
12. 确认无误后 Merge（你们不审核，也可以自己合并）。
13. 合并后：git checkout main && git pull。

方式 B（不推荐）：本地合并到 main 再 push

```
git checkout main
git pull
git merge feat/xxx
git push
```

8. 冲突处理（一定会遇到，照步骤做）

冲突的本质：你和别人同时改了同一个文件的同一段代码。解决方式是手动选择保留哪一部分。

8.1 用 VS Code 解决冲突（推荐）

14. 执行 merge 后出现冲突，VS Code 会提示哪些文件冲突。
15. 打开冲突文件，看到 <<<<<<<、=======、>>>>>>> 标记。
16. 用 VS Code 提供的按钮选择 Accept Current / Accept Incoming / Accept Both，或手动编辑。
17. 保存文件后，确保项目能运行（至少启动后端/前端不报错）。
18. 把解决后的文件 add 并 commit，再 push。

冲突解决后需要提交一次

```
git add <冲突文件>
git commit -m "chore: resolve merge conflict"
git push
```

9. main 被推坏了怎么办（不做保护必须有应急方案）

如果 main 分支突然跑不起来，优先用 revert 回滚（不重写历史）。只有在彻底救不回来时才用 reset --hard + force。

9.1 推荐：revert 回滚（更安全）

```
git checkout main
git pull
git revert <坏提交的 hash>
git push
```

9.2 兜底：强制回到某个正常提交（谨慎使用）

注意：这会重写 main 历史，所有成员需要重新同步。仅在演示前紧急救火时使用。

```
git checkout main
git pull
git reset --hard <最后一次正常提交的 hash>
git push --force
```

10. 项目内约定（减少互相踩坑）

10.1 后端文件改动约定

- 路由只写参数校验与调用服务：backend/app/api/routes/*.py
- 业务逻辑集中在 services：backend/app/services/*.py
- 数据库连接与模型集中在 db：backend/app/db/*.py
- 避免多人同时改 main.py；需要改时在群里说一声

10.2 前端文件改动约定（Vue）

- 页面放在 frontend/src/pages/（或 views/，组内统一）
- 接口请求统一封装在 frontend/src/api/（不要到处散写 fetch/axios）
- 全局配置（baseURL）从 .env 读取，不要写死

10.3 合并前自测清单（30 秒）

- 后端能启动：uvicorn 不报错，/docs 打开正常
- 前端能启动：npm run dev 不报错，页面能打开
- 新增/修改的接口在 /docs 可见，并能返回示例数据
- 没有把 .env、node_modules、照片、zip、数据库文件提交进 Git

11. 常见问题速查（新手最常见的坑）

Q：push 要求输入密码但失败

A：GitHub 不再支持账号密码推送。请在 VS Code 里重新登录 GitHub，或用浏览器授权。

Q：Permission denied (publickey)

A：你用 SSH 克隆但没配 SSH key。新手建议用 HTTPS 重新 clone。

Q：Your branch is behind origin/main

A：先 git checkout main && git pull，再把 main merge 到你的分支。

Q：不小心提交了大文件（照片/zip/db）

A：立刻撤回提交：git reset --soft HEAD~1，补 .gitignore，再重新 commit。

Q：Docker 起不来

A：检查 Docker Desktop 是否启动、是否开启 WSL2；端口 3306 是否被占用。

附录 A：建议分工（4 人）

- A：扫描与 EXIF 解析（文件遍历、RAW/JPG 配对、复制整理）
- B：数据库与统计接口（MySQL 表、SQLAlchemy、统计 API）
- C：前端页面（缩略图墙、筛选、按钮、图表）
- D：AI 接口（多模态 API 封装、并发调用、结果缓存、总结生成）

附录 B：提交信息模板（建议统一）

```
feat: ...
fix: ...
chore: ...
docs: ...
```