**后台管理子系统设计报告**

**1. 引言**

* **1.1 项目背景**

海外藏中国文物平台旨在整合和展示流失海外的中国文物信息，提供文物检索、鉴赏、学术研究等服务。后台管理子系统作为平台的核心支撑部分，负责管理平台的用户、数据、内容审核、系统运维等关键任务，确保平台的稳定、安全、高效运行。

* **1.2 系统目标**

提供一个集中化、可视化的管理界面，方便管理员进行高效操作。

实现对多类用户（后台管理员、掌上博物馆用户、知识服务子系统用户）的精细化管理。

确保用户生成内容（如评论）的合规性，通过有效的审核机制维护平台环境。

保障核心业务数据（如文物信息）的完整性和准确性。

提供可靠的数据备份与恢复机制，以及全面的操作日志追溯功能。

* **1.3 主要功能概述**

本后台管理子系统主要包括以下核心功能模块：

1. **用户管理**：管理不同类型的用户信息及权限。
2. **信息审核**：审核用户提交的评论等内容，包括自动与人工审核。
3. **数据管理**：管理核心文物数据。
4. **数据备份与恢复**：支持手动及自动的数据库备份与恢复。
5. **日志管理**：记录并查询管理员的操作日志。
6. **仪表盘**：展示系统核心运营数据。

**2. 系统架构**

* **2.1 总体架构**

系统采用经典的前后端分离架构：

**前端 (CMS/frontend)**：基于 Vue.js 技术栈，负责用户界面的展示和交互逻辑。通过 HTTP RESTful API 与后端通信。

**后端 (CMS/backend)**：基于 Python Flask 框架，负责业务逻辑处理、数据持久化、API接口提供等。

* **2.2 后端技术栈概述**

**编程语言**: Python 3

**Web框架**: Flask (~2.3.3)

**ORM/数据库交互**:

Flask-SQLAlchemy (用于 AdminUser, OperationLog, BackupRecord 等模型)

原生 SQL (PyMySQL) (用于 CloudUser (移动端用户), WebUser (网页端用户), MetClear (文物数据), Comments 等)

**数据库**: MySQL

**API认证**: Flask-JWT-Extended (JWT Token)

**定时任务**: APScheduler (用于自动备份)

**HTTP服务器 (开发)**: Werkzeug (Flask内置)

**其他库**: Flask-CORS (跨域), Flask-Migrate (数据库迁移, 虽然项目中体现不明显但config.py有配置)

* **2.3 前端技术栈概述**

**核心框架**: Vue.js 2.x (推测版本，基于 vue-template-compiler 和项目结构)

**UI组件库**: Element UI

**状态管理**: Vuex

**路由管理**: Vue Router

**HTTP客户端**: Axios

**构建工具**: Vue CLI

**其他库**: Cookies (js-cookie), NProgress (页面加载进度条)

**3. 模块化功能设计**

* **3.1 用户管理模块**

系统支持对三类用户的管理：后台管理员、掌上博物馆App用户（移动端用户）、知识服务子系统用户（网页端用户）。

* **3.1.1 管理员用户管理 (AdminUser)**

**功能**: 增、删、改、查后台管理员账户。支持分页、按用户名/邮箱搜索。

**权限**: 管理员分为不同角色（如 super\_admin, admin），不同角色拥有不同操作权限。通过 @role\_required 装饰器实现。

**数据存储**: admin\_users 表，通过 SQLAlchemy 模型 AdminUser 操作。

* **3.1.2 掌上博物馆用户管理 (CloudUser/mobile\_users)**

**功能**: 增、删、改、查移动端用户信息。支持分页、按用户名/手机号/状态搜索。可修改用户状态（如启用/禁用）。

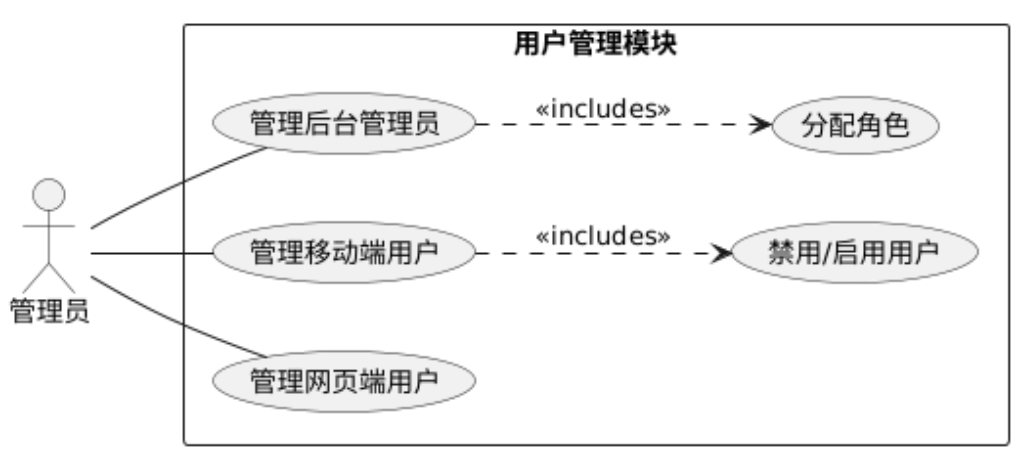
**数据存储**: mobile\_users 表，通过原生SQL和 CloudUser 类中的静态方法操作。

* **3.1.3 知识服务子系统用户管理 (WebUser/web\_users)**

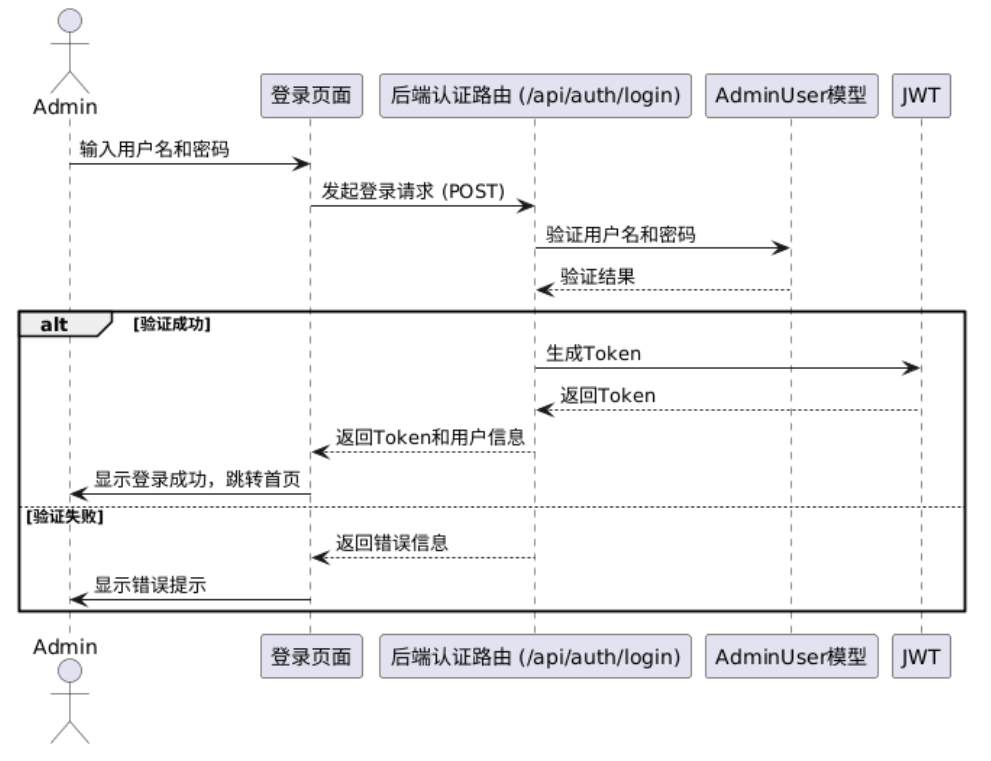
**功能**: 与移动端用户管理类似，针对网页端用户。

**数据存储**: web\_users 表，通过原生SQL和 WebUser 类中的静态方法操作。

* **3.1.4 PlantUML: 用户管理用例图**



* **3.1.5用户登录序列图 (管理员登录为例)**



* **3.2 信息审核模块**

主要针对用户发表的评论进行审核。

* **3.2.1 评论审核**

**功能**: 查询待审核/已审核评论列表（支持分页、按内容/用户/状态搜索）。管理员可对评论进行“通过”或“不通过”操作。对于不通过的评论，可同时禁用发表该评论的用户。

**数据存储**: comments 表，通过原生SQL操作。用户状态变更会更新 mobile\_users 表。

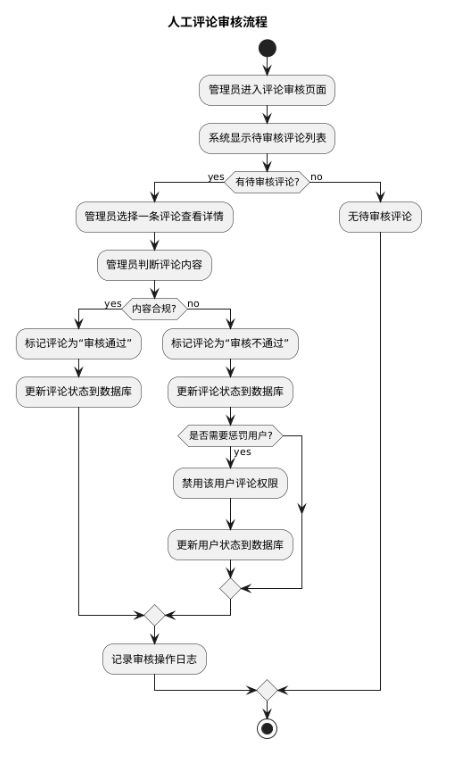
* **3.2.2 敏感词自动过滤**

**实现**: app/utils/moderation.py 中定义了 check\_sensitive\_words 函数。

**逻辑**: 该函数会加载一个敏感词列表（默认为 DEFAULT\_SENSITIVE\_WORDS，并可从数据库或文件动态加载），检查输入文本是否包含敏感词。

**应用**: 在 app/routes/review.py 的 add\_comment\_review\_log\_and\_update\_user (实际是后端提交评论的接口，但看代码更像是审核通过时调用的辅助函数) 和 update\_comment\_status (审核评论状态时) 中被间接或直接调用，用于判断是否屏蔽内容或惩罚用户。 *（实际应用点需要进一步确认，从代码看，check\_sensitive\_words 更像是提供给审核接口的一个工具，但没有在评论提交时自动拦截的明确逻辑）*

* **3.2.3 评论审核流程活动图 (人工审核)**



* **3.3 数据管理模块 (文物数据 - met\_clear)**

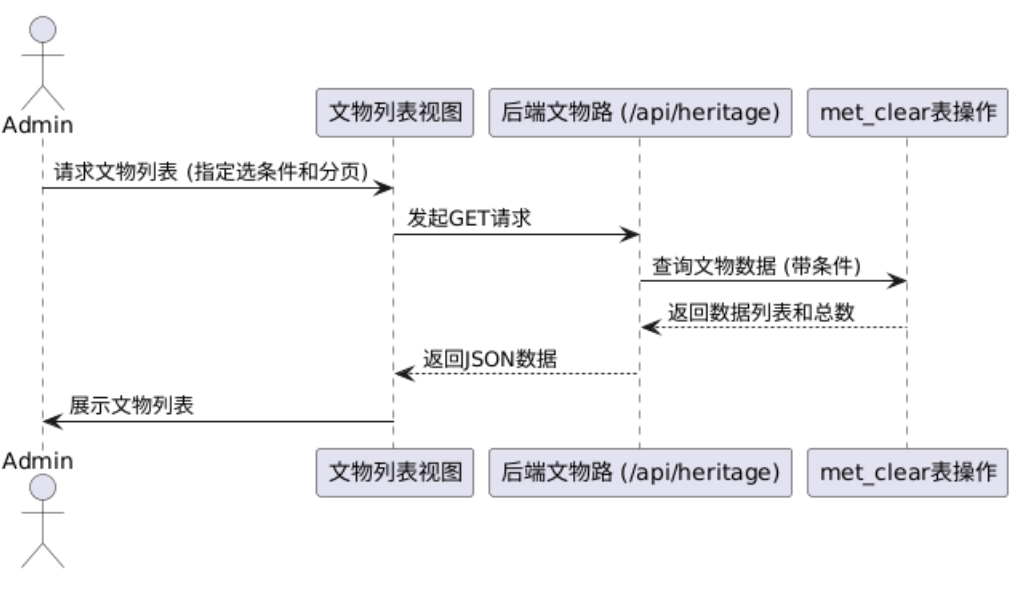
管理平台的核心业务数据——文物信息。

* **3.3.1 文物信息管理**

**功能**: 增、删、改、查文物信息。支持分页、按标题/艺术家/分类等条件搜索。支持批量删除。

**数据存储**: met\_clear 表（推测为清洗后的元数据表），通过原生SQL操作。

* **3.3.2文物数据管理序列图 (获取列表)**



* **3.4 数据备份与恢复模块**

保障系统数据的安全性和可恢复性。

* **3.4.1 手动备份与恢复**

**功能**: 管理员可手动触发数据库备份操作。可查看备份记录列表，并选择某个备份文件进行数据库恢复。

**实现**: app/utils/backup.py 中的 create\_backup 和 restore\_backup 函数，使用 mysqldump 和 mysql 命令行工具。

* **3.4.2 自动定时备份**

**功能**: 系统每日凌晨3点自动执行数据库备份。每周日凌晨4点清理超过30天的旧自动备份。

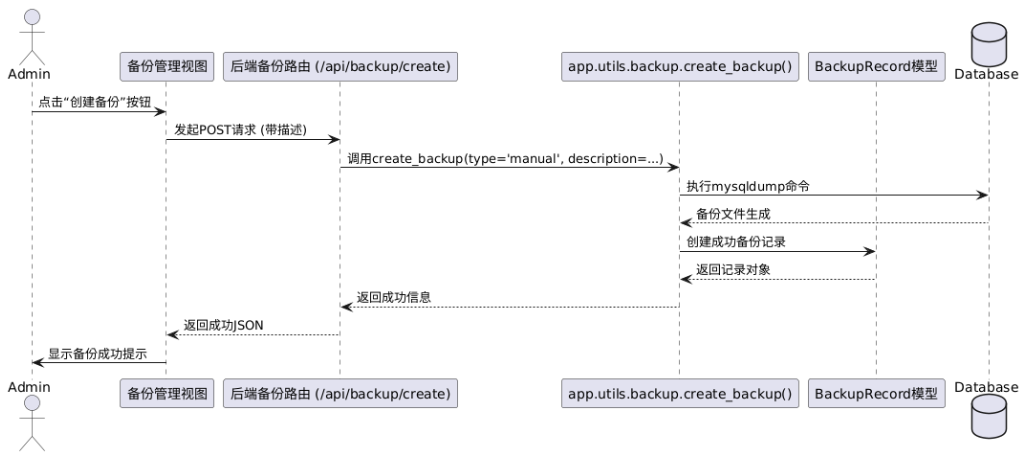
**实现**: app/utils/backup.py 中的 schedule\_backup 函数，使用 APScheduler 添加 cron 任务。

* **3.4.3 备份记录管理**

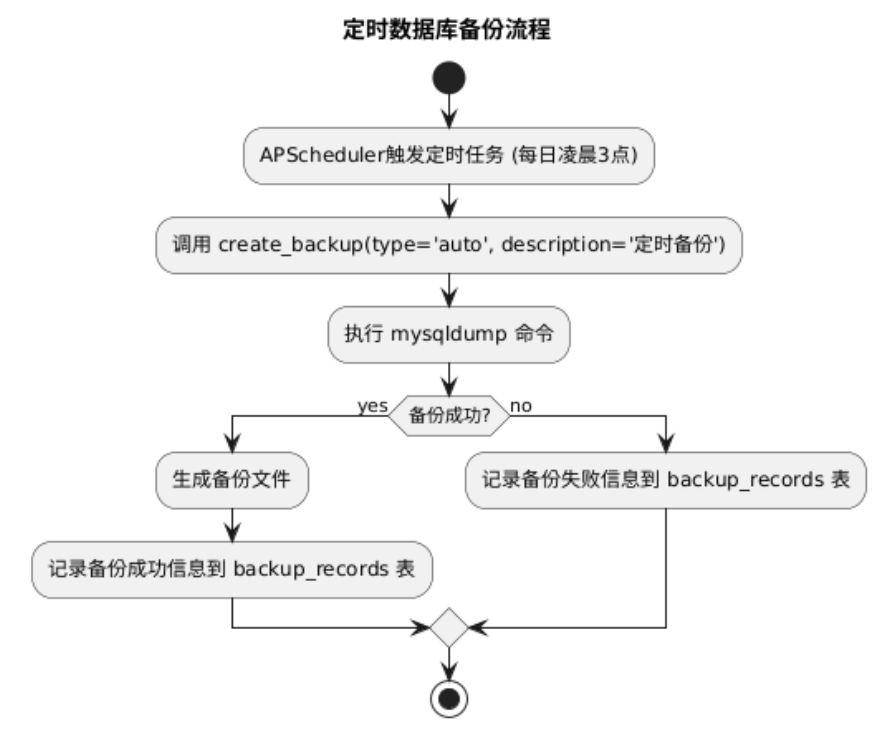
**功能**: 记录每次备份（手动/自动）的详细信息，包括备份名称、路径、大小、类型、状态、时间、描述。

**数据存储**: backup\_records 表，通过 SQLAlchemy 模型 BackupRecord 操作。

* **3.4.4手动创建备份序列图**



* **3.4.5定时备份活动图**



* **3.5 日志管理模块**

记录管理员在后台的操作，便于审计和问题追溯。

* **3.5.1 操作日志记录**

**功能**: 自动记录管理员的关键操作，如登录、用户管理操作、备份操作、审核操作等。

**记录内容**: 操作ID、管理员ID、管理员用户名、操作类型、操作具体内容、操作时间、IP地址。

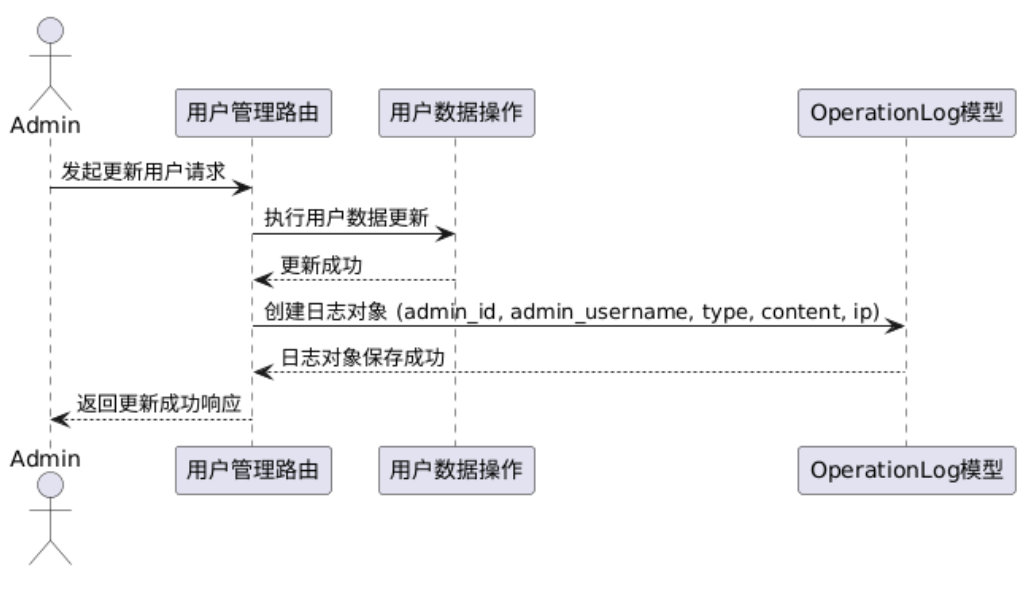
**实现**: 在各个业务路由的关键操作后，创建 OperationLog 对象并保存到数据库。

**数据存储**: operation\_logs 表，通过 SQLAlchemy 模型 OperationLog 操作。

* **3.5.2 日志查询与导出**

**功能**: 管理员可按管理员用户名、操作类型、时间范围、关键词查询操作日志。支持分页显示。支持将查询结果导出为CSV文件。

* **3.5.3操作日志记录序列图 (简化 - 以更新用户为例)**



* **3.6 仪表盘模块**

为管理员提供系统核心运营数据的概览。

* **3.6.1 核心数据统计与展示**

**功能**: 展示当日/本周/本月/总日志数，各类型日志统计，用户总数（管理员、移动端、网页端），文物总数，数据库备份数量，待审核评论数等。

**实现**: 后端 app/routes/dashboard.py 提供 /api/dashboard/stats 接口，聚合查询各类数据。

**4. 数据库设计**

* **4.1 主要数据表及其关系**

1. admin\_users: 后台管理员用户表 (SQLAlchemy管理)
2. mobile\_users: 移动端用户表 (原生SQL管理)
3. web\_users: 网页端用户表 (原生SQL管理)
4. met\_clear: 文物数据表 (原生SQL管理)
5. comments: 评论表 (原生SQL管理)

关联 mobile\_users (或 web\_users) 的用户ID。

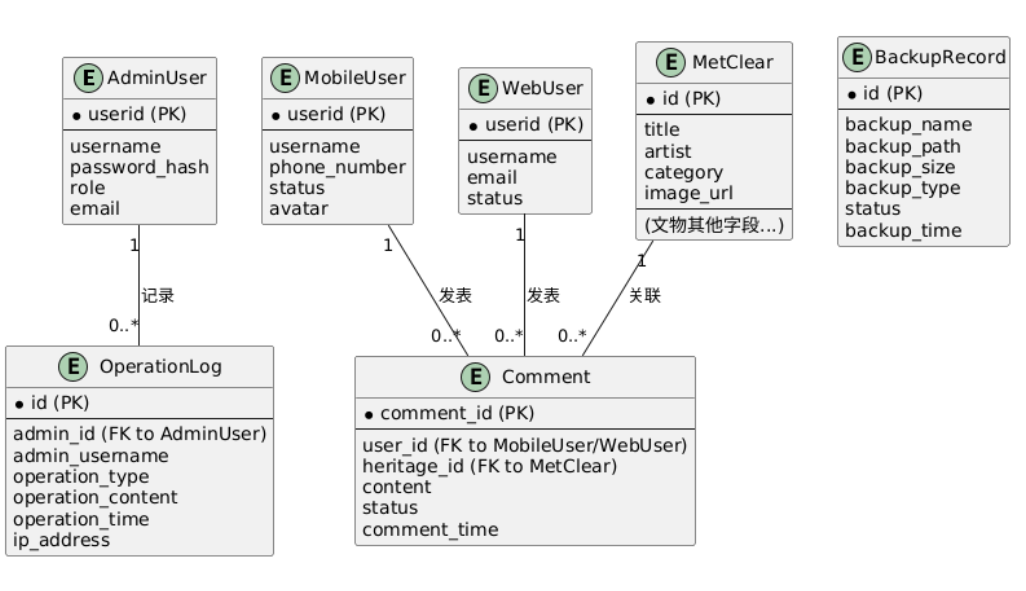
可能关联 met\_clear 的文物ID。

1. operation\_logs: 操作日志表 (SQLAlchemy管理)

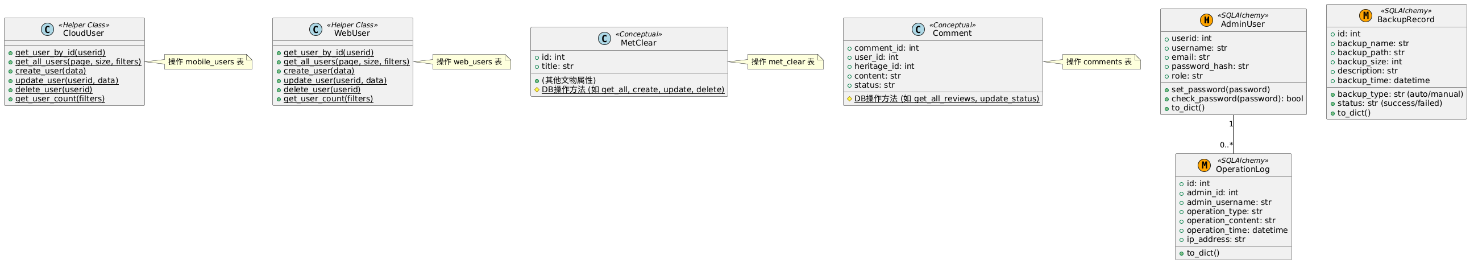
关联 admin\_users 的管理员ID。

1. backup\_records: 备份记录表 (SQLAlchemy管理)

* **4.2 简化的数据库E-R图 (核心表)**



* **4.3核心模型类图 (SQLAlchemy及概念模型)**



**5. 关键技术点实现**

* **5.1 用户认证与授权**

**认证**: 使用 Flask-JWT-Extended 实现。用户登录成功后，后端生成 JWT Token 返回给前端。前端在后续请求的 Authorization 头部携带此 Token (Bearer <token>)。

**授权**:

@jwt\_required(): Flask-JWT-Extended 提供的装饰器，用于保护需要登录才能访问的接口。

@role\_required([...roles]): 自定义装饰器 (位于 app/routes/user.py)，用于实现基于角色的访问控制。它首先通过 @jwt\_required 获取当前用户信息，然后检查用户的 role 字段是否在允许的角色列表中。

* **5.2 API 设计原则**

基本遵循 RESTful 设计风格，使用 HTTP 方法 (GET, POST, PUT, DELETE) 对应资源的增删改查操作。

URL 结构清晰，如 /api/users/admin, /api/heritage, /api/logs。

请求和响应数据主要使用 JSON 格式。

统一的响应格式：包含 code (业务状态码), message (提示信息), 和 data (业务数据) 字段，例如：

json

Apply to backup.py

        {

          "code": 20000, *// 成功*

          "message": "获取成功",

          "data": { */\* ... \*/* }

        }

        {

          "code": 50000, *// 失败*

          "message": "操作失败: 详情...",

          "data": null

        }

* **5.3 错误处理与日志记录**

**错误处理**:

在 Flask 路由函数中使用 try...except 块捕获异常。

对于业务逻辑错误或验证失败，返回包含特定 code 和 message 的 JSON 响应。

对于未捕获的服务器内部错误，Flask 会返回 500 错误。

**日志记录**:

Flask 应用本身会记录一些基本的请求和错误日志。

业务操作日志通过 OperationLog 模型显式记录到数据库。

在关键代码段（如备份工具）使用 current\_app.logger.info() 或 print() (开发阶段) 输出调试信息。

**6. 总结与展望**

**总结**: 本后台管理子系统基于 Flask 和 Vue.js 技术栈，实现了用户管理、信息审核、数据管理、备份恢复和日志管理等核心功能。系统架构清晰，模块化程度较好，基本满足了海外藏中国文物平台的后台运营需求。在数据一致性方面，部分核心数据（如用户、文物）采用原生SQL操作，而部分辅助功能（如管理员、日志、备份记录）采用SQLAlchemy ORM，这种混合方式在开发效率和灵活性上有所取舍。

**展望与建议**:

**统一数据访问层**: 考虑将所有数据库操作统一到 SQLAlchemy ORM，以提高代码一致性、可维护性，并更好地利用ORM特性（如关系管理、事务）。

**增强自动化测试**: 补充单元测试和集成测试，确保代码质量和系统稳定性。

**敏感词库管理**: 提供后台界面管理敏感词库，而非硬编码或仅通过文件配置。

**详细的API文档**: 使用 Swagger/OpenAPI 等工具生成和维护API文档。

**安全性加固**: 对SQL注入、XSS等常见Web漏洞进行更全面的审查和防护。

**前端组件复用**：检查前端视图代码(如 AdminUserManage, MobileUserManage, WebUserManage 非常相似)，考虑抽象出可复用的表格和表单组件以减少代码冗余。