智慧幕墙系统架构运维与数据集管理平台：系统需求规约 (SRS) 文档

💻 **小组成员：**2250758 林继申、2251730 刘淑仪、2256225 中谷天音

1 引言

1.1 项目背景

- **项目名称**：智慧幕墙系统架构运维与数据集管理平台
- **项目目标**：
  - 搭建智慧幕墙后端集成系统架构；
  - 构建一个自动化集成与部署的 CI/CD 流水线；
  - 进行智慧幕墙后端 GPU 算力服务器的运维工作；
  - 构建一个高效、安全、可扩展的数据集管理平台，支持数据集的上传、下载、管理、权限控制等功能。

1.2 数据集管理

1. **确保数据的完整性和一致性**：智慧幕墙系统需要处理大量实时或历史数据（如传感器数据、环境数据、用户输入等）。良好的数据集管理可以确保这些数据的完整性，防止数据丢失或冗余，同时保持数据的一致性，避免因数据冲突导致系统错误。
2. **支持数据驱动的决策**：智慧幕墙依赖于数据分析和算法来优化功能，例如玻璃幕墙平整度、石材幕墙污渍检测系统。高质量的数据集管理可以帮助系统构建可靠的训练和测试数据集，从而提升模型的准确性和实用性。
3. **提高数据访问效率**：通过对数据进行有效的分类、索引和存储管理，系统可以更快地访问和处理数据。
4. **保障数据安全性和隐私**：智慧幕墙可能涉及敏感数据（如建筑物运营数据、用户行为数据等）。数据集管理通过权限控制、加密和日志记录等方式，可以确保数据在采集、传输和存储过程中的安全性，防止未授权访问和泄露。
5. **支持系统扩展性**：随着智慧幕墙系统功能和使用规模的扩大，数据量会持续增长。合理的数据集管理策略（如数据归档、分布式存储等）有助于系统在未来扩展时仍能高效运行。
6. **优化数据生命周期管理**：数据并非永久有效，部分数据随着时间的推移可能失效或不再相关。数据集管理可以实现数据的自动清理、归档和删除，避免系统因无效数据而性能下降。

1.3 服务器运维

进行服务器运维的主要目的是确保服务器及其运行的系统和服务始终处于最佳状态，为用户和业务提供可靠、高效和安全的支持。具体来说，进行服务器运维的理由包括以下几个方面：

1. **保证系统稳定性和高可用性**
   - 连续性：服务器是业务的核心支撑，良好的运维可以避免宕机，保证服务不中断。
   - 及时恢复：通过监控和备份，可以快速发现问题并进行恢复，减少对业务的影响。
2. **保证数据和系统安全**
   - 数据保护：通过备份策略、权限控制和数据加密，确保数据的完整性和安全性。
3. **提升用户体验**
   - 快速响应：运维团队能迅速解决用户反馈的问题，减少用户等待时间。
   - 保证质量：稳定、快速的服务器服务可以提升用户对业务的满意度和信任感。

通过有效的运维可以保证业务的连续性、安全性和高效性，同时降低故障和合规风险。这是现代智慧系统和业务稳定运行的必要保障。

2 系统概述

2.1 系统目标

系统旨在通过数据集管理平台和项目运维两个方面，让智慧幕墙项目能够实现数据集的高效管理与系统的稳定运维，支持项目的快速迭代和持续改进：

- 数据集管理的目标是确保数据的高效存储、安全访问、模块化管理和标准化接口，支持大规模数据集的管理和维护。
- 运维的目标是实现环境一致性、自动化部署、系统监控、资源优化和团队协作，确保系统的稳定运行和高效迭代。

2.1.1 数据集管理

1. **高效的数据存储与访问**
   - 使用阿里云对象存储（OSS）作为数据集的持久化层，确保大规模数据的高效存储和快速访问。
   - 提供清晰的上传和下载接口，支持用户便捷地管理数据集。
2. **数据安全与权限控制**
   - 通过阿里云 RAM（Resource Access Management）实现严格的访问控制，确保只有授权用户或角色能够访问和操作特定数据集。
   - 遵循最小权限原则，避免数据泄露或误操作。
3. **数据备份与恢复**
   - 定期对数据集及相关元数据进行自动化备份，确保数据的安全性和业务连续性。
   - 在数据丢失或损坏时，能够快速恢复，减少对项目的影响。
4. **数据模块化管理**
   - 按照功能或业务领域划分数据集管理模块，确保数据集管理的独立性和可维护性。
   - 通过模块化设计，便于未来扩展新的数据集管理功能。
5. **数据接口标准化**
   - 遵循 RESTful API 设计原则，提供标准化的数据上传、下载和查询接口，确保前后端交互的清晰和高效。

2.1.2 项目运维

1. **环境一致性**
   - 确保开发、测试和生产环境的一致性，避免因环境差异导致的运行错误。
   - 使用 Docker 和 Docker Compose 技术，确保在不同环境中部署的应用行为一致。
2. **自动化部署与持续集成**
   - 通过 CI/CD 流水线实现代码的自动化构建、测试和部署，减少人工操作，提高部署效率。
   - 服务器定时拉取最新镜像并自动部署，确保开发完成的功能能够快速上线。
3. **系统监控与日志管理**
   - 提供部署日志和输出日志的监控功能，便于运维人员实时掌握系统运行状态。

- 通过日志分析，快速定位和解决系统问题，提升系统的稳定性和可靠性。

4. **资源优化与扩展性**
   - 合理配置服务器资源，确保 CI/CD 流水线和 Docker Compose 部署的高效运行。
   - 设计系统时考虑未来的扩展需求，确保系统能够随着业务增长而灵活扩展。
5. **团队协作与沟通**
   - 建立清晰的沟通渠道（如微信群、飞书等），确保团队成员之间的信息流通顺畅。
   - 通过敏捷 Scrum 框架，定期进行站会和回顾，及时共享进展与问题，提升团队协作效率。
6. **文档化与知识管理**
   - 及时更新技术架构、开发流程、CI/CD 配置等方面的文档，确保团队成员能够访问到最新的工作信息。
   - 提供详细的 API 文档和操作指南，便于团队协作和后续维护。
7. **高可用性与故障恢复**
   - 通过监控和备份策略，确保系统的高可用性，避免服务中断。
   - 在系统出现故障时，能够快速恢复，减少对业务的影响。
8. **安全性保障**
   - 通过备份策略、权限控制和数据加密，确保数据的完整性和安全性。

2.2 系统功能概述

2.2.1 数据集管理功能

1. **数据存储与访问**
   - 提供高效的数据存储服务，支持大规模数据集的上传、下载和查询。
   - 通过阿里云对象存储（OSS）实现数据的持久化，确保数据的高效访问。
2. **数据安全与权限控制**
   - 使用阿里云 RAM 进行访问控制，确保只有授权用户或角色能够访问和操作特定数据集。
   - 遵循最小权限原则，避免数据泄露或误操作。
3. **数据备份与恢复**
   - 定期对数据集及相关元数据进行自动化备份，确保数据的安全性和业务连续性。
   - 在数据丢失或损坏时，能够快速恢复，减少对项目的影响。
4. **数据模块化管理**
   - 按照功能或业务领域划分数据集管理模块，确保数据集管理的独立性和可维护性。
   - 支持未来扩展新的数据集管理功能。

5. **数据接口标准化**
    - 提供标准化的 RESTful API 接口，支持数据的上传、下载和查询操作。
    - 确保前后端交互的清晰和高效。

2.2.2 项目运维功能

1. **环境一致性**
    - 使用 Docker 和 Docker Compose 技术，确保开发、测试和生产环境的一致性，避免因环境差异导致的运行错误。
2. **自动化部署与持续集成**
    - 引入 CI/CD 流水线，实现代码的自动化构建、测试和部署，减少人工操作，提高部署效率。
    - 服务器定时拉取最新镜像并自动部署，确保开发完成的功能能够快速上线。
3. **系统监控与日志管理**
    - 提供部署日志和输出日志的监控功能，便于运维人员实时掌握系统运行状态。
    - 通过日志分析，快速定位和解决系统问题，提升系统的稳定性和可靠性。
4. **资源优化与扩展性**
    - 合理配置服务器资源，确保 CI/CD 流水线和 Docker Compose 部署的高效运行。
    - 设计系统时考虑未来的扩展需求，确保系统能够随着业务增长而灵活扩展。
5. **团队协作与沟通**
    - 建立清晰的沟通渠道（如微信群、飞书等），确保团队成员之间的信息流通顺畅。
    - 通过敏捷 Scrum 框架，定期进行站会和回顾，及时共享进展与问题，提升团队协作效率。
6. **文档化与知识管理**
    - 及时更新技术架构、开发流程、CI/CD 配置等方面的文档，确保团队成员能够访问到最新的工作信息。
    - 提供详细的 API 文档和操作指南，便于团队协作和后续维护。

2.3 系统架构概述

智慧幕墙数据集管理与运维系统采用分层架构设计，主要分为前端层、后端层和持久化层，具体架构如下：

2.3.1 前端层

- **功能**：负责用户界面的展示和用户操作的交互。
- **技术栈**：基于 Vue.js 框架，采用组件化设计，确保界面的模块化和可复用性。
- **主要模块**：

- 文件列表展示
- 文件上传与下载操作
- 用户登录与权限管理
- 系统监控与日志查看

2.3.2 后端层

- **功能**：处理业务逻辑、数据管理和权限控制，为前端提供统一的 API 接口。
- **技术栈**：基于 Spring Boot 框架，采用模块化设计，确保业务逻辑的独立性和可维护性。
- **主要模块**：
  - 用户管理：负责用户认证和权限控制。
  - 数据集管理：处理数据的上传、下载、查询等操作。
  - 日志管理：记录系统运行日志和部署日志。
  - API 网关：提供统一的 RESTful API 接口，供前端调用。

2.3.3 持久化层

- **功能**：负责数据集的存储和管理，确保数据的高效访问和安全性。
- **技术栈**：使用阿里云对象存储（OSS）作为数据存储服务，支持大规模数据的高效存储和快速访问。
- **主要功能**：
  - 数据存储：通过 OSS 实现数据集的持久化存储。
  - 数据备份：定期对数据集进行自动化备份，确保数据安全。
  - 数据恢复：在数据丢失或损坏时，能够快速恢复。

2.3.4 安全性设计

- **访问控制**：通过阿里云 RAM 实现严格的权限管理，确保只有授权用户或角色能够访问特定数据集。
- **数据加密**：对敏感数据进行加密存储，确保数据的安全性。

2.3.5 运维支持

- **环境一致性**：使用 Docker 和 Docker Compose 技术，确保开发、测试和生产环境的一致性。
- **自动化部署**：通过 CI/CD 流水线实现代码的自动化构建、测试和部署。
- **资源监控**：实时监控服务器资源使用情况，确保系统的高效运行。

3 功能需求

3.1 数据集管理功能

3.1.1 用例图分析

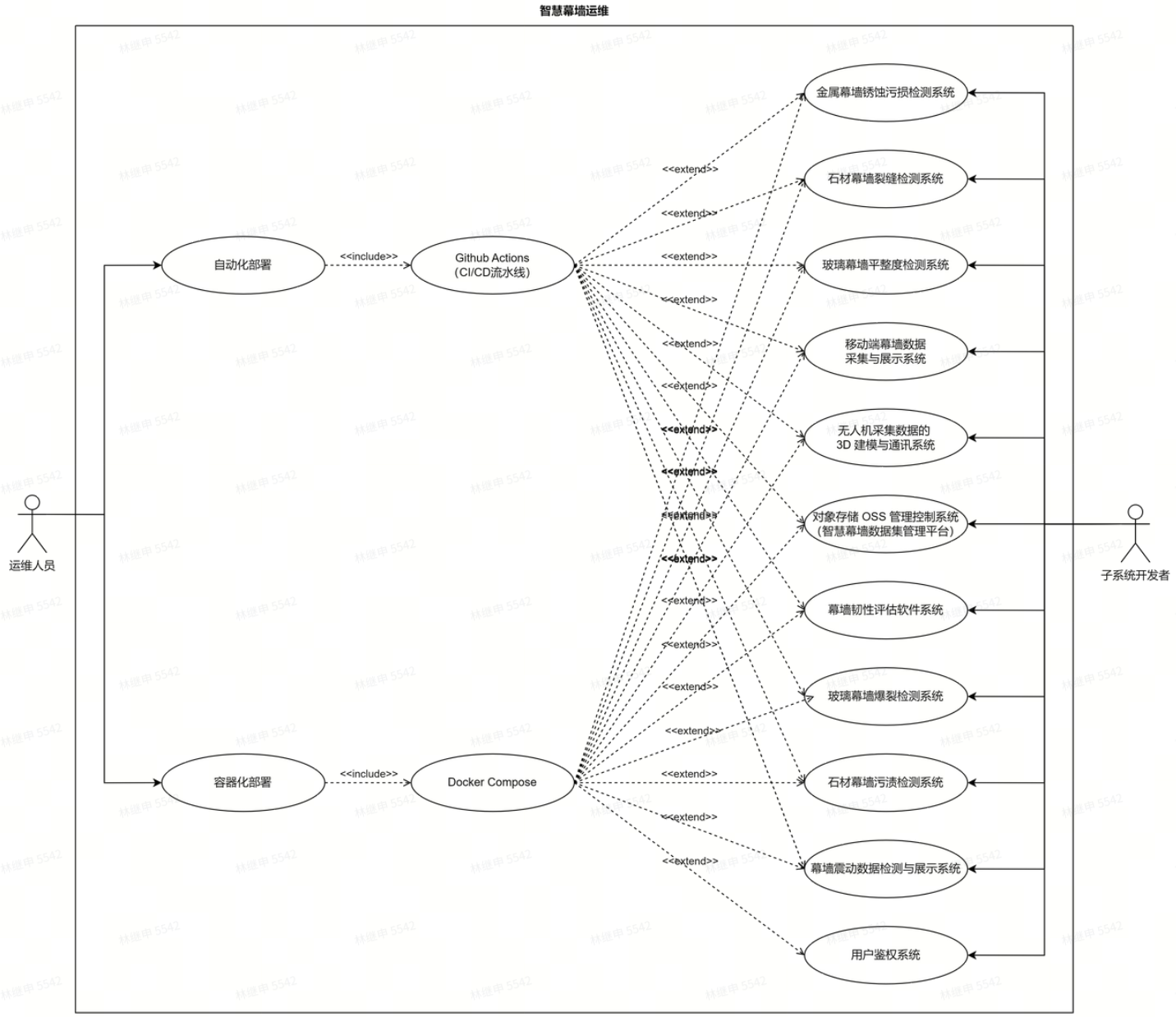

图 3-1 智慧幕墙系统部署与运维 UML 用例图

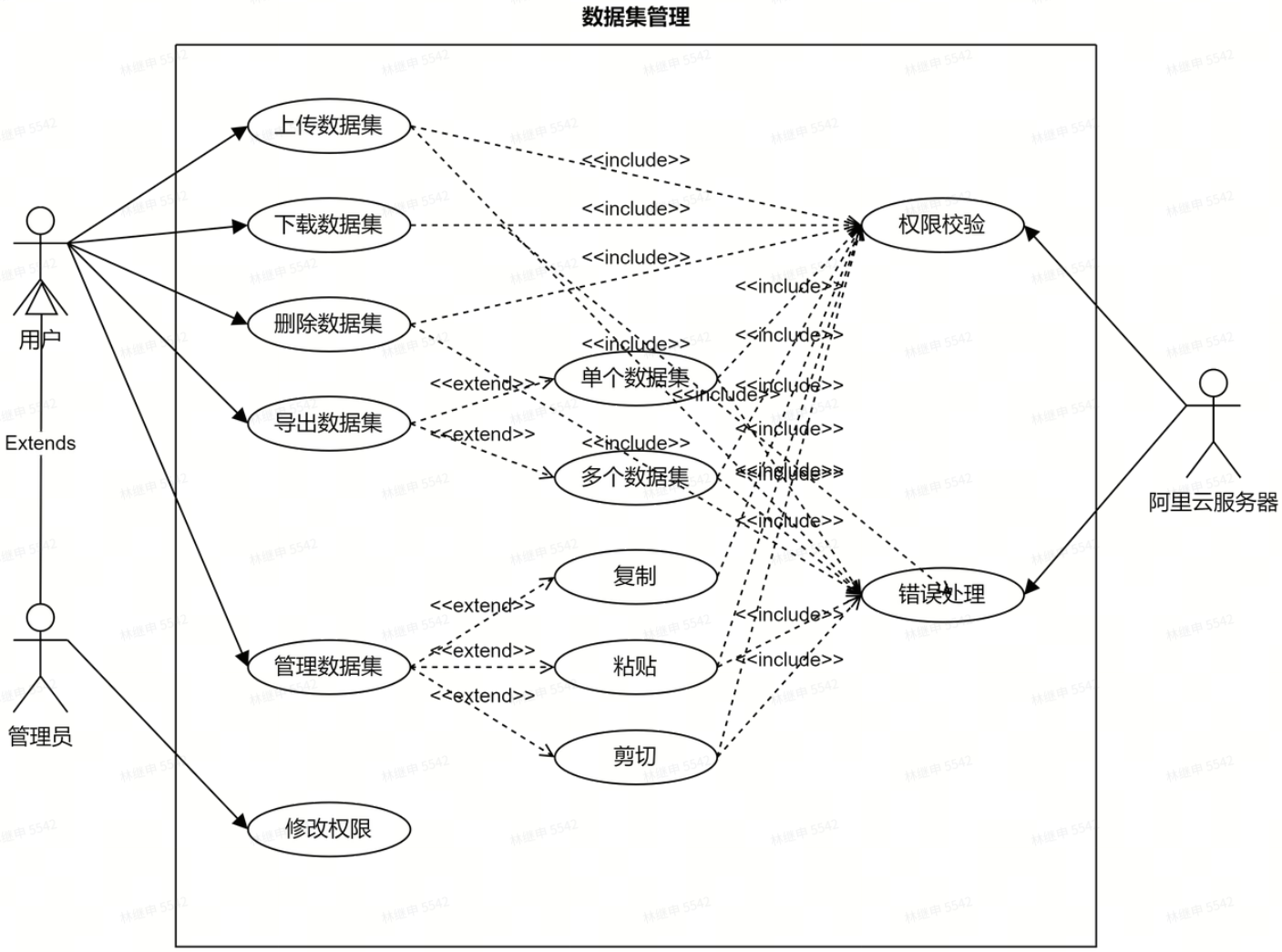

图 3-2 智慧幕墙数据集管理 UML 用例图

数据集管理功能模块共包含 7 个主要功能，涵盖了数据的上传、下载、删除、导出、管理以及权限控制等操作。每个功能都通过标准化的 RESTful API 接口与前端交互，确保数据的高效管理和安全访问。

3.1.2 活动图分析

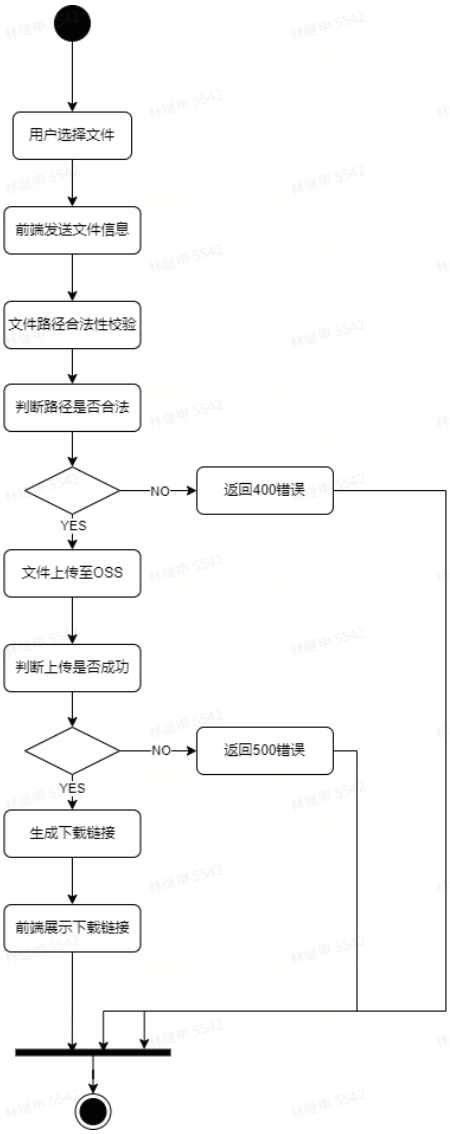

图 3-3 上传数据集 UML 活动图

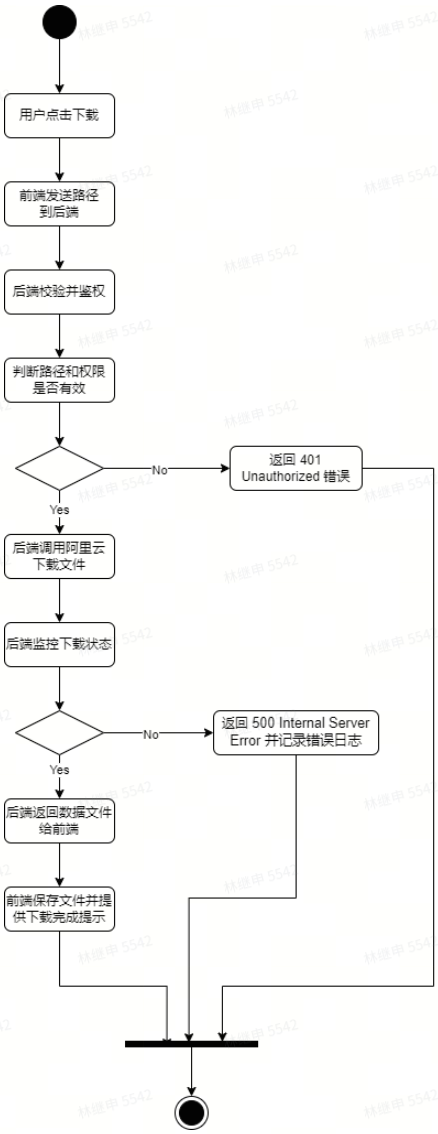

图 3-4 下载数据集 UML 活动图

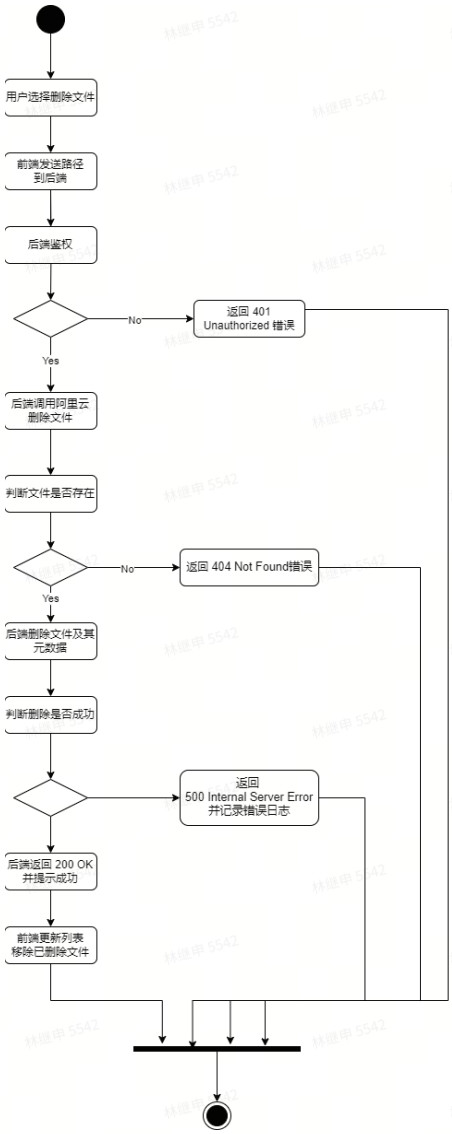

图 3-5 删除数据集 UML 活动图

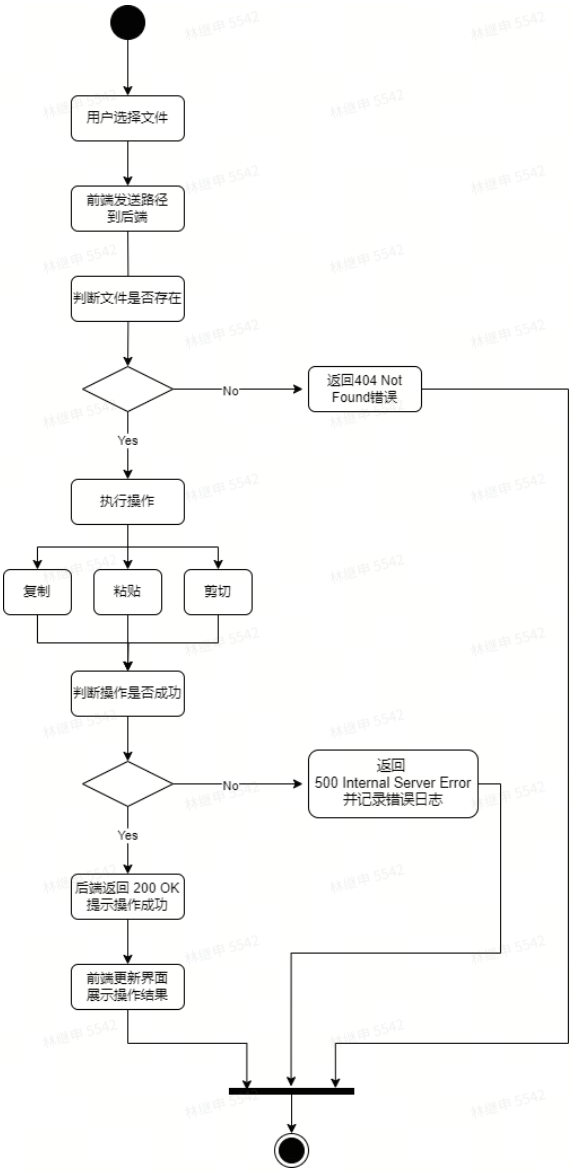

图 3-6 管理数据集 UML 活动图

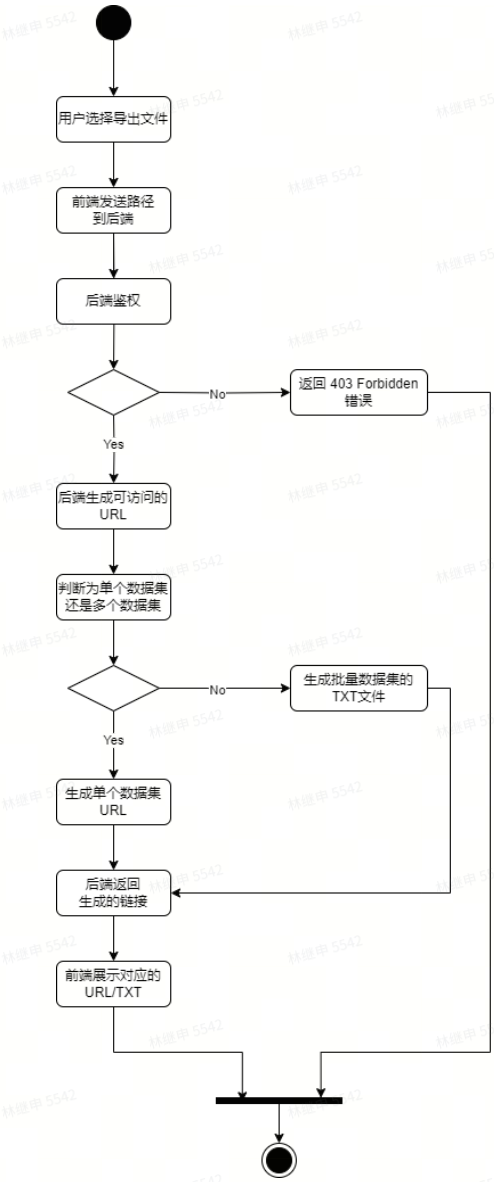

图 3-7 导出数据集 UML 活动图

3.1.3 上传数据集

1. **文件选择与上传**
   - 用户通过前端界面选择本地文件，并触发上传操作。支持单个文件上传，也支持多个文件上传。
   - 前端将文件及相关信息（如文件路径、用户信息）通过 RESTful API 发送至后端。
2. **文件路径合法性校验**
   - 后端对文件路径进行合法性校验，确保路径符合命名规则且不包含非法字符。
   - 如果路径非法，返回 400 Bad Request 错误，并提示用户修改路径。
3. **文件上传至 OSS**
   - 后端调用阿里云 OSS 的 SDK，将文件上传至指定的 OSS 存储桶。
   - 上传过程中，后端会监控上传状态，确保文件完整上传。
4. **返回下载链接**

- 文件上传成功后，后端生成文件的下载链接，并通过 API 返回给前端。
- 前端展示下载链接，用户可以通过该链接直接下载文件。

5. **错误处理**
   - 如果上传过程中出现错误（如网络中断、OSS 服务不可用），后端返回 500 Internal Server Error，并记录错误日志。
   - 前端根据错误码提示用户重新上传或联系管理员。

3.1.4 下载数据集

1. **文件路径请求**
   - 用户通过前端界面输入文件路径，并触发下载操作。
   - 前端将文件路径通过 RESTful API 发送至后端。
2. **文件路径校验与权限验证**
   - 后端对文件路径进行校验，确保路径合法且文件存在。
   - 同时，后端验证用户是否有权限下载该文件。如果无权限，返回 401 Unauthorized 错误。
3. **从 OSS 下载文件**
   - 后端调用阿里云 OSS 的 SDK，根据文件路径从 OSS 存储桶中下载文件。
   - 下载过程中，后端会监控下载状态，确保文件完整下载。
4. **返回文件数据**
   - 文件下载成功后，后端将文件的二进制数据通过 API 返回给前端。
   - 前端将文件数据保存到用户本地，并提供下载完成的提示。
5. **错误处理**
   - 如果文件不存在，后端返回 404 Not Found 错误。
   - 如果下载过程中出现错误，后端返回 500 Internal Server Error，并记录错误日志。

3.1.5 删除数据集

1. **文件选择与删除请求**
   - 用户通过前端界面选择要删除的文件，并触发删除操作。
   - 前端将文件路径及相关信息通过 RESTful API 发送至后端。
2. **权限校验**
   - 后端验证用户是否有权限删除该文件。如果无权限，返回 401 Unauthorized 错误。
3. **从 OSS 删除文件**
   - 后端调用阿里云 OSS 的 SDK，根据文件路径从 OSS 存储桶中删除文件。

- 删除操作会同步删除文件的元数据信息。

4. **返回删除结果**
   - 文件删除成功后，后端返回 200 OK 状态码，并提示用户删除成功。
   - 前端更新文件列表，移除已删除的文件。
5. **错误处理**
   - 如果文件不存在，后端返回 404 Not Found 错误。
   - 如果删除过程中出现错误，后端返回 500 Internal Server Error，并记录错误日志。

3.1.6 导出数据集

1. **单个数据集导出**
   - 用户可以选择单个数据集，并导出其对应的 URL 链接。
   - 系统会生成一个可访问的 URL 链接，用户可以通过该链接直接访问或下载数据集。
2. **批量数据集导出**
   - 用户可以选择多个数据集，批量导出其对应的 URL 链接。
   - 系统会生成一个包含所有选中数据集 URL 链接的文件（txt 文件），供用户下载和使用。
   - 批量导出的 URL 链接同样具有时效性，确保数据的安全性。
3. **权限校验**
   - 在导出数据集时，系统会进行权限校验，确保只有具有相应权限的用户才能执行导出操作。
   - 如果用户权限不足，系统会返回错误提示（如 403 Forbidden）。
4. **日志记录**
   - 每次导出操作都会被记录到操作日志中，包括导出时间、导出用户、导出的数据集信息等，便于后续审计和追踪。
5. **接口设计**
   - 提供标准化的 RESTful API 接口，支持前端调用导出功能。
   - 接口设计遵循清晰、高效的原则，确保用户能够便捷地使用导出功能。

3.1.7 管理数据集

1. **文件复制**
   - **用户操作**：用户通过前端界面选择文件，并触发复制操作。
   - **后端处理**：
     - 前端将文件路径及相关信息通过 RESTful API 发送至后端。
     - 后端调用阿里云 OSS 的 SDK，复制指定文件到目标路径。

        - 复制成功后，后端返回 200 OK 状态码，并提示用户复制成功。
    - **错误处理**：
        - 如果文件不存在，后端返回 404 Not Found 错误。
        - 如果复制过程中出现错误，后端返回 500 Internal Server Error，并记录错误日志。
2. **文件剪切**
    - **用户操作**：用户通过前端界面选择文件，并触发剪切操作。
    - **后端处理**：
        - 前端将文件路径及相关信息通过 RESTful API 发送至后端。
        - 后端调用阿里云 OSS 的 SDK，将文件从原路径移动到目标路径。
        - 剪切成功后，后端返回 200 OK 状态码，并提示用户剪切成功。
    - **错误处理**：
        - 如果文件不存在，后端返回 404 Not Found 错误。
        - 如果剪切过程中出现错误，后端返回 500 Internal Server Error，并记录错误日志。
3. **文件粘贴**
    - **用户操作**：用户通过前端界面选择一个或多个文件，并触发粘贴操作。
    - **后端处理**：
        - 前端将文件路径及相关信息通过 RESTful API 发送至后端。
        - 后端调用阿里云 OSS 的 SDK，批量粘贴文件到目标路径。
        - 粘贴成功后，后端返回 200 OK 状态码，并提示用户粘贴成功。
    - **错误处理**：
        - 如果文件不存在，后端返回 404 Not Found 错误。
        - 如果粘贴过程中出现错误，后端返回 500 Internal Server Error，并记录错误日志。

3.1.8 权限控制功能

3.1.8.1 用户角色管理

1. **用户角色定义**
    - 系统支持多种用户角色，主要包括：
        - **管理员**：拥有最高权限，可以管理所有数据集，包括上传、下载、删除、查询等操作，同时可以管理其他用户角色和权限。
        - **用户**：拥有部分权限，通常只能上传、下载和查询自己上传的数据集，无法删除他人上传的数据集。

2. **角色权限分配**
    - 每个角色在系统中都有明确的权限定义，权限包括：
        - **上传权限**：允许用户上传数据集至 OSS。
        - **下载权限**：允许用户从 OSS 下载数据集。
        - **删除权限**：允许用户删除数据集。
        - **查询权限**：允许用户查询数据集列表。
    - 管理员可以通过后台管理系统为不同角色分配或修改权限。
3. **角色权限的持久化**
    - 角色及其权限信息存储在配置文件中，确保系统重启后权限配置不会丢失。

3.1.8.2 权限校验

1. **权限校验流程**
    - 当用户尝试执行某个操作（如上传、下载、删除）时，系统会首先进行权限校验。
    - 校验流程如下：
        i. 系统根据用户的角色，查询其拥有的权限列表。
        ii. 检查当前操作是否在用户的权限范围内。
        iii. 如果用户拥有相应权限，则允许执行操作；否则，返回 401 Unauthorized 错误。
2. **上传权限校验**
    - 用户尝试上传数据集时，系统会检查其是否拥有上传权限。
    - 如果用户角色为管理员或当前权限下的用户，则可以上传。否则不予点击。
3. **下载权限校验**
    - 用户尝试下载数据集时，系统会检查其是否拥有下载权限。
    - 只有管理员或当前权限下的用户才可以下载当前数据集。
4. **删除权限校验**
    - 用户尝试删除数据集时，系统会检查其是否拥有删除权限。
    - 只有管理员或当前权限下的用户可以删除数据集，其他用户无法执行删除操作。
5. **查询权限校验**
    - 用户尝试查询数据集列表时，系统会检查其是否拥有查询权限。
    - 只有管理员或当前权限下的用户可以查询数据集列表。
6. **权限校验的实现**
    - 权限校验逻辑通过后端 Spring Boot 的拦截器（Interceptor）实现。

- 每次请求到达后端时，系统会根据用户角色和请求类型自动进行权限校验。

7. **错误处理**
   - 如果用户尝试执行超出其权限范围的操作，系统会返回 401 Unauthorized 错误，并提示用户“无权限执行此操作”。
   - 错误信息会记录到系统日志中，便于管理员排查问题。

3.2 备份与恢复功能

3.2.1 用例图分析

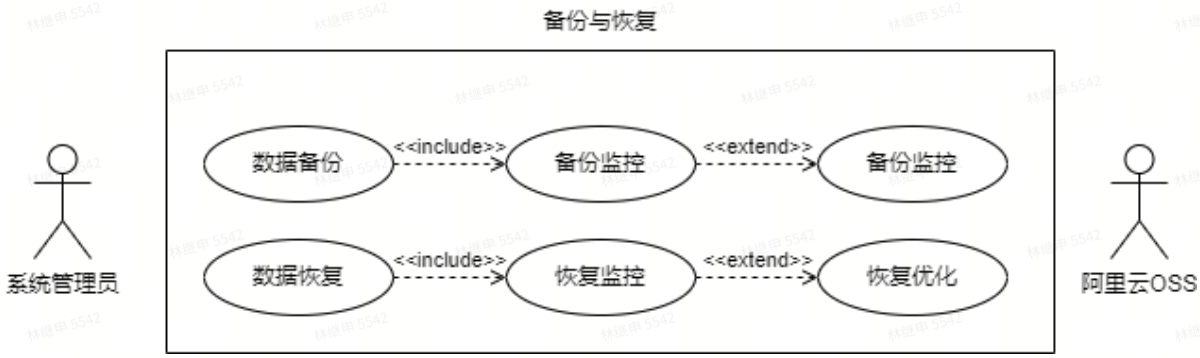

图 3-8 数据集备份与恢复 UML 用例图

备份与恢复功能模块共包含 2 个主要功能，确保数据的安全性和业务连续性。

3.2.2 数据备份

1. **备份策略**
   - **备份频率**：每周一 00:00 自动执行一次全量备份。
   - **保留时间**：每个备份保留 30 天，过期备份自动删除。
   - **备份内容**：包括所有数据集及其元数据（如文件路径、上传时间、用户信息等）。
2. **备份实现**
   - **备份工具**：使用阿里云 OSS 的备份服务，通过 SDK 或 API 将数据上传至 OSS。
   - **备份流程**：
     i. 系统在备份时间点自动触发备份任务。
     ii. 将数据集及其元数据打包为压缩文件。
     iii. 将压缩文件上传至阿里云 OSS 的指定存储桶。
     iv. 记录备份任务的执行状态和日志。
3. **备份存储**
   - 备份数据存储在阿里云 OSS 中，确保数据的高可用性和安全性。
   - 备份文件按日期命名，便于管理和查询。

4. **备份监控**
   - 系统记录每次备份的执行状态（成功/失败）和详细信息（如备份时间、文件大小）。
   - 如果备份失败，系统会发送告警通知给系统管理员，并记录错误日志。
5. **备份优化**
   - 采用增量备份与全量备份结合的方式，减少备份时间和存储空间占用。
   - 定期清理过期备份，释放存储资源。

3.2.3 数据恢复

1. **恢复场景**
   - **数据丢失**：由于误操作、硬件故障等原因导致数据集丢失。
   - **数据损坏**：数据集因病毒攻击、存储介质损坏等原因无法正常使用。
2. **恢复流程**
   - **选择备份点**：系统管理员从备份列表中选择需要恢复的备份点（如最近一次备份或特定日期的备份）。
   - **触发恢复**：系统管理员通过界面或命令行工具触发恢复操作。
   - **下载备份文件**：系统从阿里云 OSS 下载选定的备份文件。
   - **解压与恢复**：将备份文件解压，并将数据集及其元数据恢复到系统中。
   - **验证恢复结果**：系统检查恢复后的数据是否完整和可用，并记录恢复日志。
3. **恢复实现**：
   - **恢复工具**：使用阿里云 OSS 的 SDK 或 API 下载备份文件，并通过脚本或工具实现数据恢复。
   - **恢复权限**：只有系统管理员可以触发数据恢复操作，确保恢复过程的安全性。
4. **恢复监控**：
   - 系统记录每次恢复操作的执行状态（成功/失败）和详细信息（如恢复时间、恢复文件数量）。
   - 如果恢复失败，系统会发送告警通知给系统管理员，并记录错误日志。
5. **恢复优化**：
   - 提供部分恢复功能，允许系统管理员选择特定文件或目录进行恢复，减少恢复时间和资源占用。
   - 支持恢复预览功能，系统管理员可以在恢复前查看备份文件的内容，确保恢复的准确性。

3.3 日志管理功能

3.3.1 用例图分析

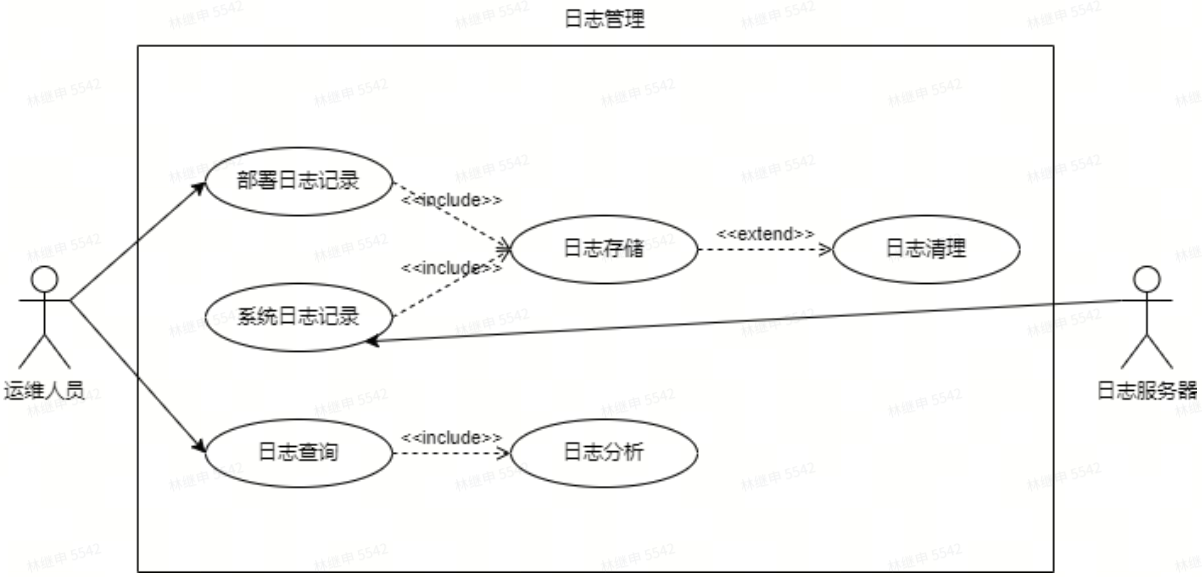

图 3-9 智慧幕墙部署与运维日志管理 UML 用例图

日志管理功能模块共包含 3 个主要功能，确保系统的可追溯性和可维护性。

3.3.2 操作日志记录

1. **日志内容**
   - **部署日志**
     - 记录每次部署的详细信息，包括：
       - 部署时间
       - 部署人员
       - 部署版本
       - 部署状态（成功/失败）
       - 部署过程中执行的命令
       - 部署失败时的错误信息
   - **系统日志**
     - 记录系统运行过程中的关键事件，如：
       - 服务启动/停止
       - 资源使用情况（CPU、内存、磁盘）
       - 异常错误（如网络中断、服务崩溃）
2. **日志格式**
   - 日志采用结构化格式，便于解析和查询。
   - 每条日志包含以下字段：
     - 时间戳

- 日志级别
- 操作类型（如部署、启动、停止）
- 操作详情
- 操作结果

3. **日志存储**
   - 日志存储在专门的日志服务器或云存储服务（如阿里云日志服务）中，确保日志的安全性和可扩展性。
   - 日志按日期或项目进行分片存储，便于管理和查询。
4. **日志记录实现**
   - 通过日志框架在代码中嵌入日志记录逻辑。
   - 部署脚本中增加日志记录功能，确保每次部署的详细操作都被记录。

3.3.3 日志查询与分析

1. **日志查询**
   - 运维人员和系统管理员可以通过日志查询界面或命令行工具查询日志。
   - 支持按时间范围、日志级别、操作类型等条件进行筛选和查询。
2. **日志分析**
   - 对日志数据进行分析，生成报告或告警，帮助运维人员监控系统健康状态。
   - 支持对异常错误进行统计和分析，帮助快速定位问题。

3.3.4 日志清理

1. **定期清理**
   - 系统定期清理过期日志，释放存储资源。
   - 清理策略可以根据日志的存储时间和存储空间进行配置。

4 非功能需求

4.1 性能需求

1. **高并发支持**
   - 系统应支持大规模数据集的上传和下载，确保在高并发情况下的性能稳定。
   - 在高并发场景下，系统的响应时间应保持在合理范围内（如上传和下载操作的响应时间不超过 5 秒）。
2. **响应时间优化**

- 上传和下载操作的响应时间应控制在合理范围内，确保用户体验。
- 对于小于 1GB 的文件，上传和下载操作的响应时间应不超过 3 秒；对于大于 1GB 的文件，响应时间应不超过 10 秒。

3. **资源利用率**
   - 系统应优化服务器资源（如 CPU、内存、磁盘 I/O）的使用，确保在高负载情况下仍能高效运行。
   - 系统应支持动态资源分配，根据负载情况自动调整资源使用。
4. **缓存机制**
   - 系统应引入缓存机制（如 Redis），减少对数据库和存储服务的频繁访问，提升数据读取速度。
   - 对于频繁访问的数据集，系统应提供缓存支持，减少重复计算和查询的开销。

4.2 安全性需求

1. **权限控制**
   - 系统应具备严格的权限控制机制，确保只有符合权限要求的用户才能访问和操作数据。
   - 权限控制应基于角色（如管理员、普通用户、访客），并支持细粒度的权限分配（如按数据集、按操作类型）。
2. **数据加密**
   - 系统应具备数据加密机制，确保数据在存储和传输过程中的安全性。
   - 敏感数据（如用户信息、数据集元数据）应进行加密存储，防止未授权访问。
3. **传输安全**
   - 系统应使用 HTTPS 协议进行数据传输，确保数据在传输过程中不被窃取或篡改。
   - 对于敏感操作（如上传、下载、删除），系统应进行双重验证（如密码、短信验证码）。
4. **日志与审计**
   - 系统应记录所有关键操作（如上传、下载、删除、权限变更）的日志，便于审计和追踪。
   - 日志应包含操作时间、操作用户、操作类型、操作结果等详细信息。
5. **防攻击机制**
   - 系统应具备防攻击机制，如防止 SQL 注入、XSS 攻击、DDoS 攻击等。
   - 系统应定期进行安全漏洞扫描和修复，确保系统的安全性。

4.3 可扩展性需求

1. **模块化设计**

- 系统应采用模块化设计，确保各功能模块（如数据集管理、权限控制、日志管理）的独立性和可扩展性。
- 新增功能模块时，应尽量减少对现有系统的影响。

2. **分布式支持**
   - 系统应支持分布式部署，能够根据业务需求扩展服务器节点，提升系统的处理能力。
   - 系统应支持负载均衡，确保在高并发情况下仍能稳定运行。
3. **存储扩展**
   - 系统应支持存储扩展，能够根据数据量的增长动态扩展存储资源（如阿里云 OSS 的存储容量）。
   - 系统应支持多种存储服务（如本地存储、云存储），便于根据需求选择合适的存储方案。
4. **接口扩展**
   - 系统应提供标准化的 RESTful API 接口，便于与其他系统进行集成。
   - 新增接口时，应确保与现有接口的兼容性，避免对现有功能造成影响。

4.4 可维护性需求

1. **代码结构**
   - 系统应具备良好的代码结构，遵循高内聚低耦合的设计原则，便于理解和维护。
   - 代码应遵循统一的编码规范，确保代码的一致性和可读性。
2. **文档化管理**
   - 系统应提供详细的文档，包括技术架构、开发流程、API 文档、操作指南等，便于团队协作和后续维护。
   - 文档应及时更新，确保与系统实际功能保持一致。
3. **日志与监控**
   - 系统应提供详细的日志记录和监控功能，便于运维人员实时掌握系统运行状态。
   - 日志应包含系统运行的关键信息（如错误日志、性能日志），便于问题排查和性能优化。
4. **自动化测试**
   - 系统应支持自动化测试，确保新增功能或修改不会影响现有功能的正常运行。
   - 自动化测试应覆盖核心功能（如上传、下载、删除、权限控制），确保系统的稳定性。
5. **版本管理**
   - 系统应使用版本管理工具（如 Git），确保代码的可追溯性和可回滚性。
   - 每次发布新版本时，应记录版本变更内容，便于后续维护和问题排查。

5 系统设计原则

5.1 高内聚低耦合

1. **高内聚**
   - 模块内部的功能应紧密相关，职责单一，确保每个模块只负责一个特定的功能或业务逻辑。
   - 例如，在 Spring Boot 中，一个 Service 类应专注于处理特定的业务逻辑，而不是混杂多个职责。
   - 高内聚的设计便于代码的理解、维护和测试，减少模块内部的复杂性。
2. **低耦合**
   - 模块之间的依赖关系应尽量减少，通过接口或事件机制进行通信，避免模块之间的直接依赖。
   - 例如，前端 Vue 组件之间通过 Vuex 状态管理进行通信，而不是直接相互引用。
   - 低耦合的设计便于模块的独立开发、测试和替换，提升系统的灵活性和可扩展性。

5.2 错误处理机制

1. **错误反馈**
   - 系统应具备有效的错误处理机制，确保在出现错误时能够及时反馈给用户，提供清晰的错误提示。
   - 例如，上传文件失败时，系统应返回具体的错误信息（如“文件路径非法”或“网络中断”），帮助用户快速定位问题。
2. **错误日志**
   - 所有错误应被记录到系统日志中，包括错误时间、错误类型、错误详情等信息，便于后续排查和分析。
   - 错误日志应存储在专门的日志服务器或云存储服务中，确保日志的安全性和可追溯性。
3. **错误恢复**
   - 系统应具备一定的错误恢复能力，能够在出现错误时自动尝试恢复操作（如重试上传或下载）。
   - 对于无法自动恢复的错误，系统应提供手动恢复的选项（如重新上传文件或联系管理员）。

5.3 模块化原则

1. **前端模块化**
   - 前端采用 Vue.js 框架，利用其组件化特性，将界面拆分为独立、可复用的组件，提升开发效率和代码可维护性。
   - 使用 Vue Router 进行页面路由管理，确保各模块之间的独立性。

2. **后端模块化**
   - 后端采用 Spring Boot 框架，按照功能或业务领域划分模块（如用户管理、数据集管理、权限控制等），每个模块独立开发和维护。
   - 模块之间通过接口或事件机制进行通信，避免直接依赖。
3. **模块复用**
   - 模块化设计应支持功能的复用，减少重复代码的开发。
   - 例如，权限控制模块可以在多个子系统中复用，避免重复实现相同的功能。

5.4 RESTful API 设计

1. **标准 HTTP 方法**
   - 系统接口应遵循 RESTful 设计原则，使用标准的 HTTP 方法（如 GET、POST、PUT、DELETE）对应相关操作。
   - 例如，GET 用于查询数据，POST 用于创建数据，PUT 用于更新数据，DELETE 用于删除数据。
2. **清晰的 URL 结构**
   - URL 结构应反映资源之间的关系，便于理解和使用。
   - 例如，`/oss/upload/{文件路径}` 用于上传文件，`/oss/download/{文件路径}` 用于下载文件。
3. **状态码与响应体**
   - 接口应使用标准的 HTTP 状态码（如 200、400、401、404、500）传达操作结果，确保接口的可理解性。
   - 响应体应包含清晰的结构化数据（如 JSON 格式），便于前端解析和处理。
4. **版本控制**
   - 接口应支持版本控制，便于后续的扩展和兼容性维护。
   - 例如，`/v1/oss/upload` 表示第一版的上传接口，`/v2/oss/upload` 表示第二版的上传接口。

5.5 可扩展性和可维护性

1. **可扩展性**
   - 系统设计时应考虑未来的扩展需求，确保系统能够随着业务增长而灵活扩展。
   - 系统应支持分布式部署和负载均衡，便于在高并发情况下扩展服务器节点。
2. **可维护性**
   - 系统应具备良好的代码结构和文档化管理，便于后续的维护和升级。

- 代码应遵循统一的编码规范，确保代码的一致性和可读性。

3. **文档化**
   - 系统应提供详细的文档，包括技术架构、开发流程、API 文档、操作指南等，便于团队协作和后续维护。
   - 文档应及时更新，确保与系统实际功能保持一致。
4. **自动化测试**
   - 系统应支持自动化测试，确保新增功能或修改不会影响现有功能的正常运行。
   - 自动化测试应覆盖核心功能（如上传、下载、删除、权限控制），确保系统的稳定性。
5. **版本管理**
   - 系统应使用版本管理工具（如 Git），确保代码的可追溯性和可回滚性。
   - 每次发布新版本时，应记录版本变更内容，便于后续维护和问题排查。

6 技术选型

6.1 前端技术

1. **Vue.js**
   - **用途**：用于构建用户界面，支持组件化和状态管理。
   - **优势**：Vue.js 是一个轻量级、高性能的前端框架，具有简单易学的 API 和强大的生态系统，适合快速开发复杂的单页应用（SPA）。
   - **特性**：支持双向数据绑定、组件化开发、虚拟 DOM 等，提升开发效率和用户体验。
2. **Vue Router**
   - **用途**：用于页面路由管理，支持单页应用的多页面切换。
   - **优势**：Vue Router 是 Vue.js 的官方路由库，提供灵活的路由配置和导航控制，支持动态路由、嵌套路由等功能。
   - **特性**：支持路由懒加载，提升页面加载速度；提供路由守卫，确保页面访问的安全性。
3. **Vuex**
   - **用途**：用于状态管理，确保组件之间的通信和数据共享。
   - **优势**：Vuex 是 Vue.js 的官方状态管理库，提供集中式的状态管理，便于在复杂应用中管理共享状态。
   - **特性**：支持状态持久化、模块化管理，确保状态的可预测性和可维护性。

6.2 后端技术

1. **Spring Boot**

- **用途**：用于构建后端服务，处理业务逻辑和数据管理。
- **优势**：Spring Boot 是一个基于 Spring 框架的快速开发工具，提供自动配置和嵌入式服务器支持，简化了 Spring 应用的开发和部署。
- **特性**：支持 RESTful API、数据库访问、安全认证等功能，适合构建微服务架构。

2. **RESTful API**
   - **用途**：用于前后端交互，确保接口的清晰和高效。
   - **优势**：RESTful API 是一种基于 HTTP 协议的接口设计风格，具有简单、灵活、可扩展的特点，便于前后端分离开发。
   - **特性**：使用标准的 HTTP 方法（如 GET、POST、PUT、DELETE）和状态码（如 200、400、500），确保接口的规范性和可理解性。

6.3 部署技术

1. **Docker Compose**
   - **用途**：Docker Compose 是一种用于定义和运行多容器 Docker 应用程序的工具。它通过一个 YAML 文件（通常命名为 `docker-compose.yml`）来配置应用程序的服务、网络和卷等资源，使得开发者能够轻松地管理复杂的多容器应用。Docker Compose 特别适用于开发、测试和持续集成环境，能够快速启动和停止整个应用栈。
   - **优势**：Docker Compose 的主要优势在于其简化和标准化了多容器应用的部署流程。通过一个配置文件，开发者可以清晰地定义各个服务之间的依赖关系和配置，避免了手动启动和管理多个容器的繁琐操作。此外，Docker Compose 支持环境变量和配置文件的重用，使得在不同环境（如开发、测试、生产）之间切换变得更加容易。
   - **特性**：Docker Compose 提供了多种特性来增强容器化应用的管理能力。它支持服务依赖管理，确保容器按正确顺序启动；允许通过 `docker-compose up` 和 `docker-compose down` 命令一键启动和停止整个应用栈；还支持水平扩展服务实例。此外，Docker Compose 与 Docker Swarm 和 Kubernetes 等编排工具兼容，能够无缝集成到更复杂的部署流程中。

6.4 持久化技术

1. **阿里云 OSS 对象存储服务**
   - **用途**：用于数据集的存储和管理，确保数据的高效访问和安全性。
   - **优势**：阿里云 OSS 提供高可靠、低成本的对象存储服务，支持大规模数据的存储和访问，适合作为数据集的持久化层。
   - **特性**：支持数据备份、恢复、版本控制等功能，确保数据的完整性和可追溯性。

7 接口设计

OSS 控制器 API 提供文件的上传与下载功能，通过 HTTP 请求与服务端交互。所有接口均以 `/oss` 为基础路径。

7.1 下载文件接口

1. **接口描述**：通过提供文件的路径，从 OSS 存储中下载指定文件。
2. **URL**：`GET /oss/download/{文件路径}`
3. **请求参数**：无直接请求参数。

> ⚠ 文件的最终存储路径（包括文件名）需要在上传请求的 URL 中明确指定。举例来说，如果您上传的原始文件名是 `upload.txt`，并在请求URL中指定存储路径为 `/oss/upload/user/documents/file.txt`，那么该文件会以指定的文件名 `file.txt` 存储于 `user/documents/` 目录下，即最终存储路径为 `user/documents/file.txt`。请注意，这里的 `file.txt` 是需要由您在上传请求中明确指定的新文件名（可以与原文件名重合）。

4. **响应**：
   - **成功**：返回文件的二进制数据，同时响应头包含文件下载的必要信息。
     - **HTTP 状态码**：`200 OK`
     - **响应头**：`Content-Disposition: attachment; filename=<文件名>`
     - **响应体**：文件的二进制内容
   - **失败**：返回 HTTP 错误状态码。
     - `404 Not Found`：文件不存在。

7.2 上传文件接口

1. **接口描述**：通过提供文件及相关用户信息，将文件上传至 OSS 存储。
2. **URL**：`POST /oss/upload/{文件路径}`
3. **请求参数**：

参数名	类型	必填	描述
file	MultipartFile	是	要上传的文件
userName	String	是	账号（用于身份验证）
password	String	是	密码（用于身份验证）

⚠ 文件路径通过 URL 动态指定。例如，上传的文件为 `upload.txt`，上传路径为 `/oss/upload/user/documents/file.txt`，则 `upload.txt` 会存储为 `user/documents/file.txt`。

4. **响应**：
   - **成功**：返回文件的下载链接。
     - **HTTP 状态码**：200 OK
     - **响应体**：`{ "downloadUrl": "http://<服务域名>/oss/download/<文件路径>" }`
   - **失败**：返回 HTTP 错误状态码及错误信息。
     - `401 Unauthorized`：身份验证失败。
     - `400 Bad Request`：对象键格式无效（如路径或文件名非法）。
     - `500 Internal Server Error`：文件上传失败。
5. **验证规则**：
   - 目录部分（路径中的每一层目录）只能包含字母、数字和 `-`。
   - 文件名部分只能包含字母、数字、`.` 和 `-`。
   - 不允许使用连续的斜杠（`//`）。

8 数据流图

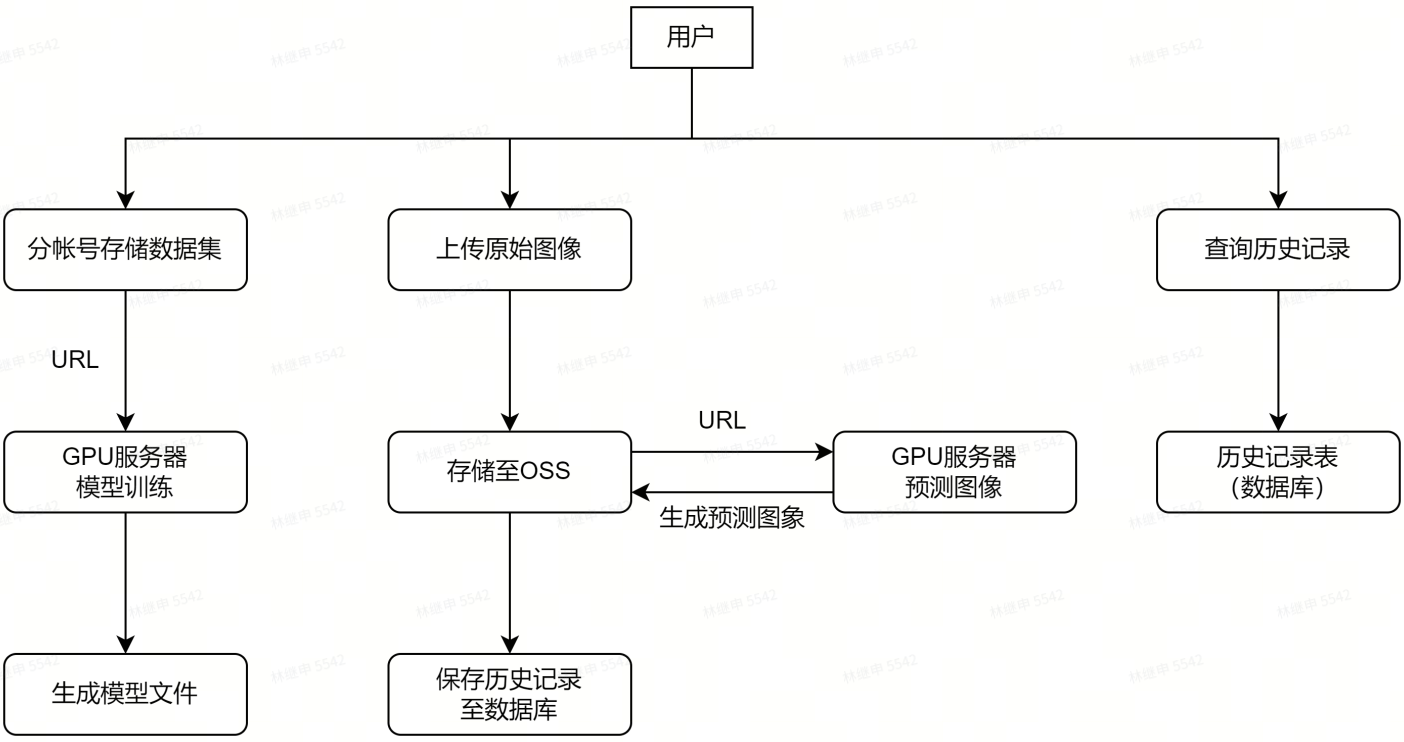

图 8-1 数据流图

1. **用户**：用户是系统的起点，负责上传原始图像和查询历史记录。

2. **分账号存储数据集**：系统支持为不同用户账号分别存储数据集，确保数据的隔离性和安全性。
3. **上传原始图像**：用户可以将原始图像上传到系统中，这些图像将用于后续的模型训练和预测。
4. **查询历史记录**：用户可以查询之前处理过的图像和预测结果的历史记录。
5. **GPU 服务器**：GPU 服务器用于执行计算密集型任务，如模型训练和图像预测。
6. **模型训练**：使用上传的原始图像在 GPU 服务器上进行模型训练，生成模型文件。
7. **存储至 OSS**：训练生成的模型文件和其他数据被存储到阿里云 OSS（对象存储服务）中，确保数据的高可用性和持久性。
8. **预测图像**：使用训练好的模型在 GPU 服务器上对图像进行预测，生成预测结果。
9. **历史记录表（数据库）**：系统将用户的操作记录和预测结果保存到数据库中，便于后续查询和分析。
10. **生成模型文件**：在模型训练过程中，生成模型文件，这些文件将用于后续的图像预测。
11. **保存历史记录至数据库**：系统将用户的操作记录和预测结果保存到数据库中，确保历史数据的可追溯性和可查询性。

9 系统展示

9.1 系统部署与运维

该图片展示了数据库运维平台的界面设计，主要包括数据库的监控、管理和维护功能。用户可以通过该界面实时查看数据库的运行状态、执行 SQL 查询、备份与恢复数据等操作。

```
(base) mat@matcloud:~/Intelligent_Curtain_Wall$ sudo docker images
REPOSITORY                              TAG                       IMAGE ID       CREATED        SIZE
minmuslin/intelligent-curtain-wall      crack-detection           fd66af3542c5   4 days ago     6.21GB
minmuslin/intelligent-curtain-wall      flatness-detection        13642aaced36   5 days ago     6.53GB
minmuslin/intelligent-curtain-wall      spalling-detection        567f27b51388   5 days ago     6.56GB
minmuslin/intelligent-curtain-wall      mobile-data               eab6de1a2973   10 days ago    469MB
minmuslin/intelligent-curtain-wall      resilience-assessment     9a491bb295cd   10 days ago    499MB
minmuslin/intelligent-curtain-wall      modeling-communication    b2f84ce55e70   10 days ago    562MB
minmuslin/intelligent-curtain-wall      corrosion-detection       86f8e819aa35   12 days ago    7.47GB
minmuslin/intelligent-curtain-wall      vibration-detection       09dfc1127635   2 weeks ago    607MB
minmuslin/intelligent-curtain-wall      stain-detection           a1c2bbed2edc   2 weeks ago    6.79GB
minmuslin/intelligent-curtain-wall      user-authentication       0fd32b7212ae   2 weeks ago    581MB
minmuslin/intelligent-curtain-wall      mobile-data-data-logger   6d6297333f7e   2 weeks ago    133MB
(base) mat@matcloud:~/Intelligent_Curtain_Wall$ sudo docker compose down
[+] Running 12/12
 ✓ Container flatness-detection              Removed
 ✓ Container crack-detection                 Removed
 ✓ Container modeling-communication          Removed
 ✓ Container user-authentication             Removed
 ✓ Container corrosion-detection             Removed
 ✓ Container resilience-assessment           Removed
 ✓ Container stain-detection                 Removed
 ✓ Container spalling-detection              Removed
 ✓ Container vibration-detection             Removed
 ✓ Container mobile-data                     Removed
 ✓ Container mobile-data-data-logger         Removed
 ✓ Network intelligent_curtain_wall_default  Removed
(base) mat@matcloud:~/Intelligent_Curtain_Wall$ sudo docker compose pull
[+] Pulling 11/11
 ✓ resilience-assessment Pulled
 ✓ modeling-communication Pulled
 ✓ vibration-detection Pulled
 ✓ corrosion-detection Pulled
 ✓ user-authentication Pulled
 ✓ flatness-detection Pulled
 ✓ mobile-data Pulled
 ✓ spalling-detection Pulled
 ✓ mobile-data-data-logger Pulled
 ✓ crack-detection Pulled
 ✓ stain-detection Pulled
(base) mat@matcloud:~/Intelligent_Curtain_Wall$ sudo docker compose up -d
[+] Running 12/12
 ✓ Network intelligent_curtain_wall_default  Created
 ✓ Container spalling-detection              Started
 ✓ Container mobile-data-data-logger         Started
 ✓ Container corrosion-detection             Started
 ✓ Container mobile-data                     Started
 ✓ Container stain-detection                 Started
 ✓ Container user-authentication             Started
 ✓ Container modeling-communication          Started
 ✓ Container resilience-assessment           Started
 ✓ Container crack-detection                 Started
 ✓ Container vibration-detection             Started
 ✓ Container flatness-detection              Started
```

图 9-1 系统部署与运维

```
data:View Ultralytics Settings with 'yolo settings' or at '/root/.config/Ultralytics/settings.json'
data:Update Settings with 'yolo settings key=value', i.e. 'yolo settings runs_dir=path/to/dir'. For help see https://docs.ultralytics.com/quickstart/#ultralytics-settings.
data: * Serving Flask app 'app'
data: * Debug mode: off
data:[31m[1mWARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.[0m
data: * Running on all addresses (0.0.0.0)
data: * Running on http://127.0.0.1:8080
data: * Running on http://172.18.0.10:8080
data:[33mPress CTRL+C to quit[0m
data:192.168.122.1 - - [07/Jan/2025 04:25:41] "[33mGET / HTTP/1.1[0m" 404 -
data:192.168.122.1 - - [07/Jan/2025 08:17:33] "POST /history HTTP/1.1" 200 -
```

图 9-2 系统日志输出

9.2 数据集管理平台

使用图形用户界面（GUI）。以下是一些具体的特征和功能：

1. **可视化元素**：平台使用了图标、按钮、菜单和列表等图形元素，使用户能够通过点击和选择来操作。
2. **文件管理**：用户可以通过界面浏览、复制、剪切、删除和粘贴文件，这些操作通常通过图形按钮或右键菜单完成。
3. **数据展示**：平台以表格或列表的形式展示文件信息，如文件名、大小、日期等，用户可以直观地查看和管理数据。
4. **导航和操作**：用户可以通过图形界面导航到不同的目录或功能模块，如数据采集、3D建模、可视化环境等。
5. **交互反馈**：平台可能提供操作反馈，如确认对话框、进度条或状态提示，以增强用户体验。
6. **多任务处理**：用户可以在同一界面中同时进行多项操作，如查看数据、上传文件、下载文件等。

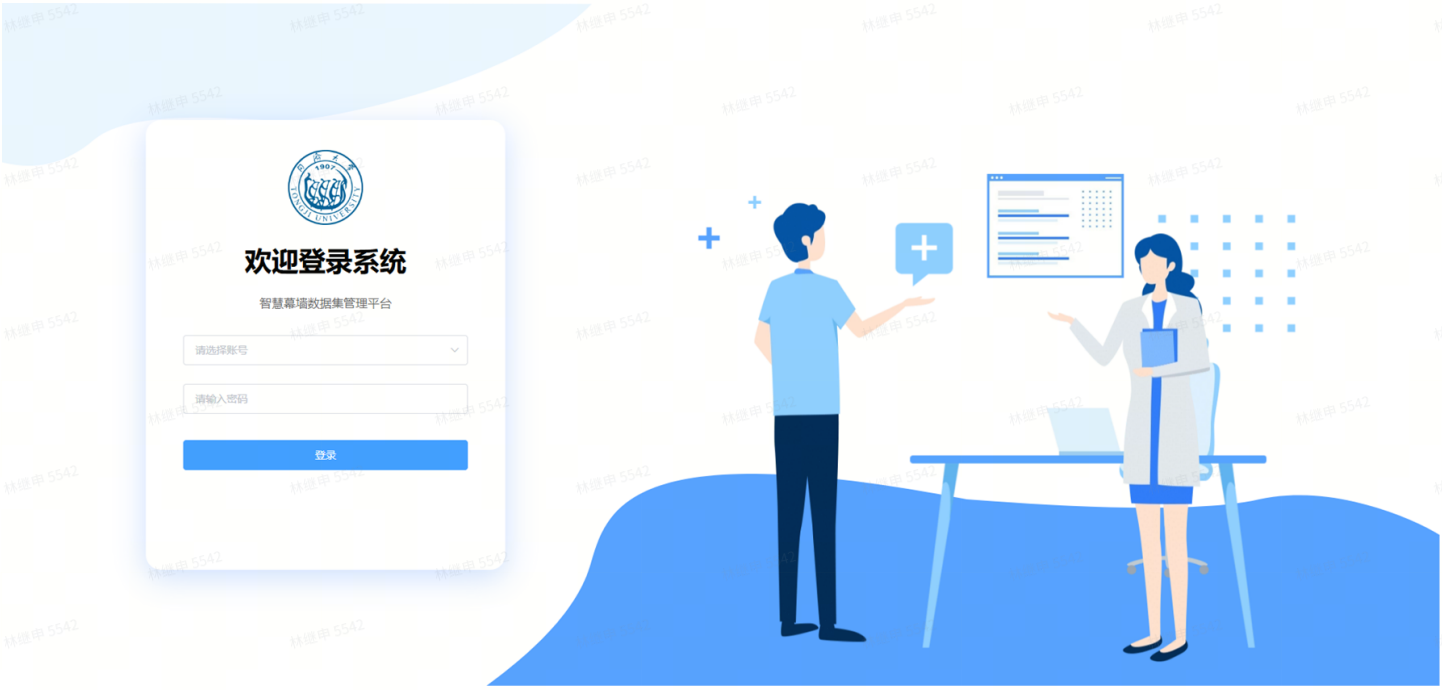

图 9-3 数据集管理平台登录页

图 9-4 数据集管理平台管理页

图 9-5 数据集管理平台控制台

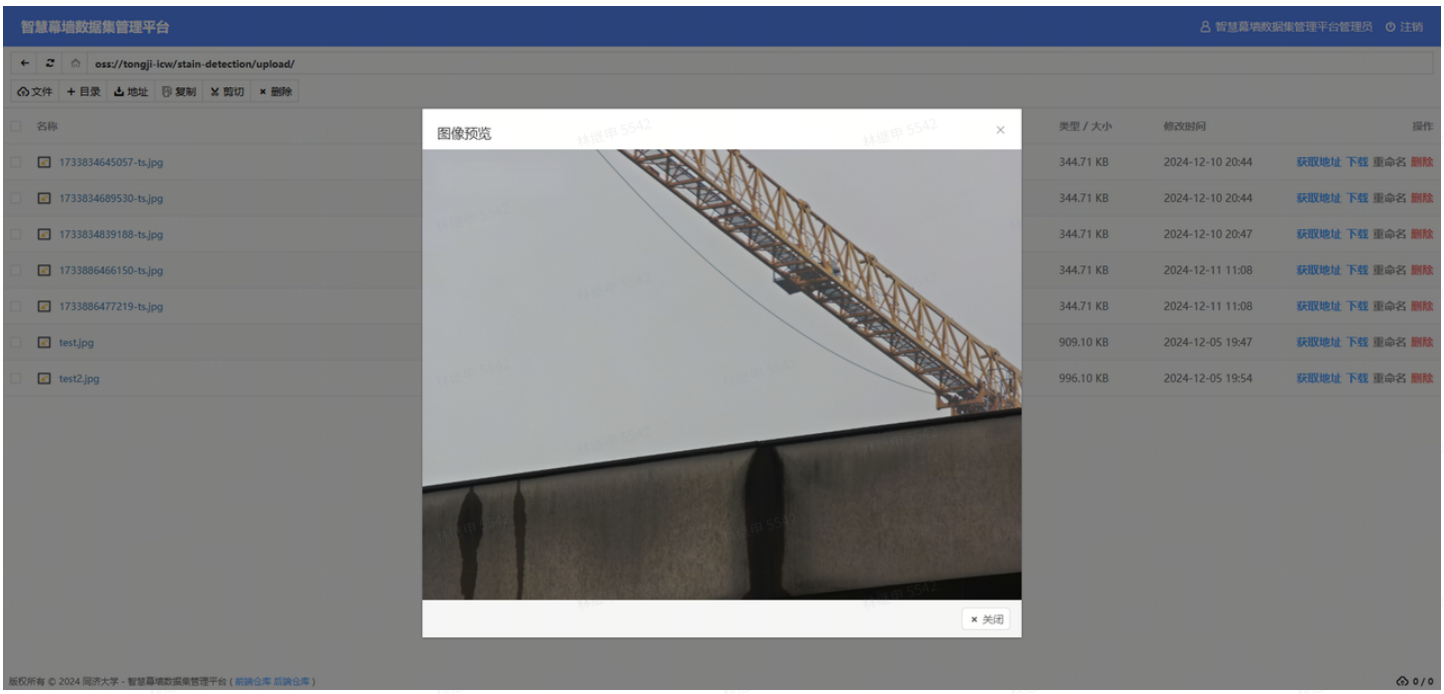

图 9-6 数据集管理平台登录页图像预览示例 1

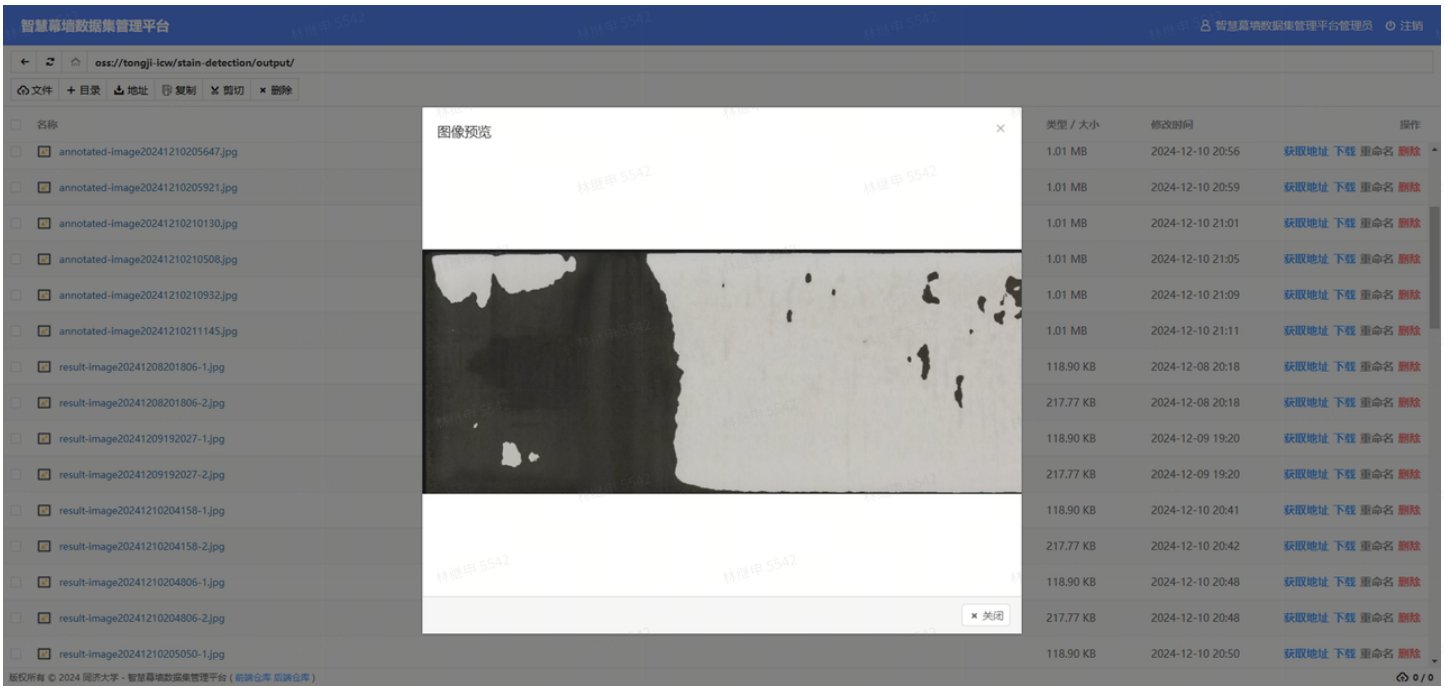

图 9-7 数据集管理平台登录页图像预览示例 2

10 总结

10.1 需求分析总结

本需求分析文档详细描述了智慧幕墙数据集管理与运维系统的功能需求和非功能需求。通过对系统的数据集管理、服务器运维、权限控制、备份与恢复、日志管理等功能的分析，明确了系统的核心目标和设计原则。

- **数据集管理**
  - **高效存储与访问**：系统使用阿里云对象存储（OSS）作为数据集的持久化层，确保大规模数据的高效存储和快速访问。
  - **数据安全与权限控制**：通过阿里云 RAM 实现严格的访问控制，确保只有授权用户或角色能够访问和操作特定数据集。
  - **数据备份与恢复**：定期对数据集进行自动化备份，确保数据的安全性和业务连续性。
  - **数据模块化管理**：按照功能或业务领域划分数据集管理模块，确保数据集管理的独立性和可维护性。
  - **数据接口标准化**：提供标准化的 RESTful API 接口，支持数据的上传、下载和查询操作。
- **项目运维**
  - **环境一致性**：使用 Docker 和 Docker Compose 技术，确保开发、测试和生产环境的一致性。
  - **自动化部署与持续集成**：引入 CI/CD 流水线，实现代码的自动化构建、测试和部署。
  - **系统监控与日志管理**：提供部署日志和输出日志的监控功能，便于运维人员实时掌握系统运行状态。
  - **资源优化与扩展性**：合理配置服务器资源，确保系统的高效运行和未来扩展。

- **团队协作与沟通**：通过敏捷 Scrum 框架，定期进行站会和回顾，提升团队协作效率。

- **非功能需求**
  - **性能需求**：系统需支持高并发访问，优化响应时间，确保用户体验。
  - **安全性需求**：系统需具备严格的权限控制、数据加密、传输安全、日志与审计、防攻击机制等安全措施。
  - **可扩展性需求**：系统需支持模块化设计、分布式部署、存储扩展和接口扩展。
  - **可维护性需求**：系统需具备良好的代码结构、文档化管理、日志与监控、自动化测试和版本管理。

10.2 未来开发计划

- **数据集管理**
  - **增强数据分类与索引功能**：进一步优化数据分类和索引机制，提升数据查询效率。
  - **支持更多数据格式**：扩展系统支持的数据格式，满足更多业务需求。
  - **数据生命周期管理**：实现数据的自动清理、归档和删除，优化数据生命周期管理。
- **项目运维**
  - **自动化运维工具**：引入更多自动化运维工具，提升运维效率。
  - **智能监控与告警**：实现智能监控和告警功能，及时发现和解决系统问题。
  - **资源动态调整**：支持服务器资源的动态调整，根据负载情况自动分配资源。
- **安全性**
  - **多因素认证**：引入多因素认证机制，提升系统安全性。
  - **安全审计**：定期进行安全审计，确保系统的安全性。
- **用户体验**
  - **用户界面优化**：持续优化用户界面，提升用户体验。
  - **用户反馈机制**：建立用户反馈机制，及时收集和处理用户意见。
- **技术升级**
  - **新技术引入**：持续关注新技术发展，适时引入新技术，提升系统性能。
  - **系统性能优化**：持续优化系统性能，确保系统在高负载情况下的稳定运行。