MailMerge - 数据汇总工具

2. 系统架构概览

2.1 后端架构 (Backend)

> 建议结合发在群里的"项目总体框架"来看，会更加清晰

后端的核心框架为 **FastAPI**：

- **技术栈**：Python 第三方库 FastAPI
- **作用**：监听所有来自前端的请求，并将后端各业务能力（数据库、任务调度、邮件服务、对象存储、LLM 代理等）统一封装为标准化的 HTTP API。通过 FastAPI 的 *router* 机制，将不同的 request 自动分发给对应的业务模块进行处理，并以 json 格式将结果返回前端。
- **位置**：`backend/api/`

主要的业务模块如下：

- **A. 数据库 (Database)**
  - **技术栈**
    - PostgreSQL 作为后端数据库
    - Python 的 SQLAlchemy 作为对象关系映射（ORM，Object-Relational Mapping）框架
  - **统一接口**
    - `get_db_session()`：用于 FastAPI 依赖注入的数据库会话提供器，会为每个请求创建一个 Session，并在请求结束后自动关闭。

      > 注意：Session 不是数据库连接，它是 ORM 的会话对象。在执行 SQL 时，Session 会临时向连接池借用数据库连接（Connection），使用后归还。
  - **位置**：`backend/database/`
- **B. 存储服务 (Storage Service)**
  - **技术栈**
    - MinIO 作为对象存储服务（兼容 S3 协议）
    - 使用 Python 第三方库 minio 与本地 MinIO 存储服务建立连接
  - **统一接口**
    - `upload(local_file_path: str, target_path: str) -> str`：上传文件到存储源指定位置
      - **本地存储**：将文件复制到本地文件系统指定路径。
      - **MinIO 存储**：将文件上传到指定桶和对象路径，若桶不存在会自动创建。
    - `download(source_path: str, local_file_path: str) -> str`：从存储源下载指定文件到本地文件系统。
    - `delete(target_path: str) -> bool`：删除存储源指定位置的文件。
    - `parse_path(path: str) -> Dict`：解析存储路径，自动判断类型（本地或 MinIO）并返回相关信息。

> 在存储服务这一块，我们统一在路径前面加上 `local://` 或 `minio://` 前缀来区分本地和 MinIO 路径，并通过 `parse_path` 对路径进行统一的格式解析

    - **位置**：`backend/storage_service/`
- **C. 邮件服务 (Email Service)**
    - **技术栈**
        - 使用 Python 内置库 `smtplib` 实现邮件发送
        - 使用 Python 内置库 `imaplib` 实现邮件接收
        - `email` 模块用于解析邮件内容及处理附件
    - **统一接口**
        - `send_email(sender_email, sender_password, receiver_email, email_content, smtp_server, smtp_port) -> Dict`：发送邮件，在 `email_content` 字段中以 `json` 格式包括邮件的标题、正文以及附件路径（如果存在）
        - `fetch_email(email_address, email_password, imap_server, imap_port, only_unread, download_dir) -> Dict`：获取邮件并解析正文和附件，并将附件下载到一个指定目录（如果没有指定，会创建一个临时目录用于保存），返回的结果中包含附件所在的路径（调用者自行处理附件）
    - **位置**：`backend/email_service/`
- **D. 调度器 (Scheduler)**
    - **技术栈**
        - 使用 APScheduler 作为后台定时任务调度框架
    - **功能**
        - 执行周期性后台任务，例如：
            - 定期检查任务状态并自动触发数据汇总逻辑
            - 定期抓取并处理邮件
        - 提供任务执行日志记录，包括执行步骤、统计信息和错误信息
        - 支持从配置文件 (`config.yaml`) 动态设置任务间隔
    - **主要接口**
        - `start_scheduler()`：启动后台调度器，根据配置添加周期性任务，并开始运行。
        - `stop_scheduler()`：停止后台调度器，安全关闭正在执行的任务。
        - 周期性任务：
            - `check_all_tasks()`：检查所有相关任务的状态（DRAFT、ACTIVE、AGGREGATED），并对满足条件的任务进行自动发布、自动关闭并汇总导出、标记需要重新汇总

              > 自动发布：对于 DRAFT 任务，当前时间 > 任务设定的发布时间
              >
              > 自动关闭并汇总导出：对于 ACTIVE 任务，当前时间 > 任务设定的截止时间
              >
              > 标记需要重新汇总：对于 AGGREGATED 任务，若该任务存在新到达的邮件，标记为 NEEDS_REAGGREGATION

            - `fetch_emails_job()`：抓取新邮件并处理附件，存储到本地数据库中
    - **位置**：`backend/scheduler/`
- **E. LLM 助手 (Agent Service)**

    - **技术栈**
        - 使用第三方库 openai 接入兼容 openai 风格的 LLM API
        - 使用阿里“通义千问-plus”作为模型
    - **功能**：通过自然语言处理用户指令，实现自动化操作，包括：
        - SQL 查询生成与执行
        - 表单模板创建
        - 邮件内容生成与发送
        - 任务创建与管理
    - **统一接口**：
        - `process_user_query(user_input: str, user_id: Optional[int] = None)`

            > `user_id` 发起本次对话的教秘 ID，用于权限检查和数据过滤
    - **运作逻辑**
        - **(1) 意图识别**：首先让 LLM 根据用户原始的 query 识别用户意图，在以下五种 ACTION 类型中选择一种：`SQL_QUERY`, `CREATE_TEMPLATE`, `CREATE_TASK`, `SEND_EMAIL`, `UNKNOWN`
        - **(2) 根据 ACTION 分发给不同的子模块处理**：每个子模块会在用户原始 query 的基础上，附加本任务所需的专有上下文信息（如任务背景、目标、限制条件），生成新的 prompt，并引导 LLM 以 json 格式 + tool call 的形式进行回应。

            不同的子模块提供如下统一的处理接口：
            - `handle_sql_query`：SQL 查询执行
            - `handle_create_template`：模板创建
            - `handle_send_email`：发送邮件
            - `handle_create_task`：任务创建
        - **(3) 执行与错误重试**：根据 LLM 返回的 tool call 执行相应逻辑的代码；若执行失败，则在 `max_try` 次以内，将「当前上下文 + 错误信息」回传给 LLM，让其重新生成修正后的 tool call 并再次执行；若给定次数内尝试均未成功，则返回错误以及失败原因。
    - **位置**：`backend/agent_service/`
- **G. 其他模块**
    - **Logger**: 统一日志管理 (`backend/logger/`)。
    - **Utils**: 通用工具函数，如统一加密处理、Excel 文件处理、统一时区的时间获取等 (`backend/utils/`)。

2.2 前端架构 (Frontend)

前端位于 `frontend/` 目录下，采用原生 Web 技术栈。

- **技术栈**: HTML5, CSS3, JavaScript (ES6+)。
- **交互方式**: 通过 JavaScript 的 `fetch` API 与后端 FastAPI 接口进行异步通信。
- **主要模块**:
    - HTML
        - **登录/注册**: `login.html`, `register.html`

        - **首页概览**: `dashboard.html` - 展示一些汇总的数据指标。
        - **表单模板**: `templates.html` - 管理数据收集的 Excel 模板。
        - **任务总览**: `tasks.html` - 管理收集任务。
        - **邮件箱**: `mailbox.html` - 以任务为模块，查看每个任务收发的邮件记录。
        - **已汇总表单**: `aggregations.html` - 管理每个任务合并并导出之后的汇总表。
        - **智能助手**: `agent.html` - 与 LLM Agent 进行对话交互。
        - **系统设置**: `settings.html` - 配置邮箱授权码等个人信息。
        - **导航栏**: `sidebar.html`：系统页面左侧的导航窗口，通过点击不同的标签跳转到不同的子界面
        - **顶部横幅**: `header.html`：系统页面最上方，横幅样式
    - CSS：`static/css/`

      > 控制页面元素的具体样式

    - JavaScript：`static/js/`

      > 控制页面元素的渲染逻辑、弹窗的跳转逻辑以及向后端发送 request 获取必要的数据等

3. 项目配置与部署

3.1 环境要求

- **操作系统**: Linux (推荐)
- **Python**: 3.11
- **Database**: PostgreSQL
- **Storage**: Docker （需要以 Docker 容器的形式运行 MinIO）

3.2 配置文件

项目依赖以下配置文件：

1. `.env` (环境变量)：用于配置敏感信息和基础设施连接。根据 `.env.example` 的示例，在项目的根目录下重新创建一个 `.env` 文件，并配置如下字段：

```
# Database Configuration
DB_HOST=localhost
DB_PORT=5432
DB_NAME=mailmerge
DB_USER=postgres
DB_PASSWORD=your_password_here

# MinIO Configuration
MINIO_ENDPOINT=localhost:9000
MINIO_ACCESS_KEY=your_access_key_here
MINIO_SECRET_KEY=your_secret_key_here
MINIO_SECURE=false
MINIO_BUCKET=mailmerge

# Local Storage Fallback
LOCAL_DATA_ROOT=your_local_data_root_here
```

```
17
18 # Security
19 SECRET_KEY=your_secret_key_here
20 ALGORITHM=HS256
21 ACCESS_TOKEN_EXPIRE_MINUTES=30
22
23 # LLM Configuration
24 DASHSCOPE_API_KEY=your_dashscope_api_key_here
25 BASE_URL=your_base_url_here
26 MODEL_NAME=qwen-plus
27 MAX_RETRY=3
28 LLM_TIMEOUT=60
29 LLM_ENABLED=true
```

配置字段	所属模块	含义
DB_HOST	Database	数据库主机地址（通常为本机或服务器 IP）
DB_PORT	Database	PostgreSQL 服务监听端口（默认 5432）
DB_NAME	Database	数据库名称
DB_USER	Database	登录数据库的用户名
DB_PASSWORD	Database	登录数据库的密码（需要自行设置）
MINIO_ENDPOINT	MinIO	MinIO 服务地址（含端口）
MINIO_ACCESS_KEY	MinIO	MinIO 的访问密钥 Access Key
MINIO_SECRET_KEY	MinIO	MinIO 的访问密钥 Secret Key
MINIO_SECURE	MinIO	是否使用 HTTPS（true/false）
MINIO_BUCKET	MinIO	默认使用的桶（Bucket）名称
LOCAL_DATA_ROOT	Local Storage	本地文件存储的根路径（当 MinIO 服务无法启动时，可以作为存储服务的替代方案）
SECRET_KEY	Security	用于 JWT 加密的密钥（必须保密）
ALGORITHM	Security	JWT 的加密算法（一般为 HS256）
ACCESS_TOKEN_EXPIRE_MINUTES	Security	访问令牌有效时长（分钟）
DASHSCOPE_API_KEY	LLM	用于调用 LLM 的 API Key
BASE_URL	LLM	模型服务的 Base URL
MODEL_NAME	LLM	默认使用的模型名称
MAX_RETRY	LLM	LLM 调用失败时最大重试次数
LLM_TIMEOUT	LLM	LLM 请求超时时间（秒）

配置字段	所属模块	含义
LLM_ENABLED	LLM	是否启用 LLM 模块（true/false）

在示例 `.env.example` 文件中，有一些字段是占位符，**部署前必须由你自行替换为真实的生产环境值。**这些字段包括：

- `DB_PASSWORD`（数据库密码）
- `MINIO_ACCESS_KEY` / `MINIO_SECRET_KEY`（MinIO 密钥）
- `LOCAL_DATA_ROOT`（本地文件存储路径）
- `SECRET_KEY`（JWT 密钥，非常重要，必须自行设置）
- `DASHSCOPE_API_KEY`（LLM API Key）
- `BASE_URL`（模型服务地址，根据你的部署环境填写）

2. `config.yaml` **（应用级配置）**：用于存放 **非敏感** 的应用运行参数，例如日志输出粒度、后台调度器的执行周期等。典型字段包括：
   - **LOG_DETAIL_LEVEL**：控制日志信息的详略程度
     - NONE：不输出调试信息
     - LOW：输出较为简单的调试信息
     - HIGH：输出更为详细的调试信息
   - **SCHEDULER.EMAIL_INTERVAL**：邮件发送调度器的运行间隔（单位：秒）。
   - **SCHEDULER.TASK_INTERVAL**：任务扫描/更新调度器的运行间隔（单位：秒）。

3.3 依赖安装

```
pip install -r requirements.txt
```

3.3 启动与管理

> 注：系统在启动时会：
>
> 1）检查数据库连接，请确保配置的 PostgreSQL 服务可以访问
>
> 2）尝试通过 Docker 启动 MinIO，请确保 Docker 服务已运行。

1. **启动应用**：项目提供了一键启动脚本 `start.sh`。
   - **前台模式**：

     ```
     ./start.sh
     ```

   - **后台模式**：

     ```
     ./start.sh --backend
     ```

     日志将输出至 `logs/service.log`，PID 记录在 `logs/service.pid`。
   - **重置所有数据文件**：

     ```
     ./start.sh --reset --set-default
     ```

- `--reset`：删除存储所有的数据库中的所有表的记录，以及 MinIO BUCKET 中的所有对象，使其恢复至初始状态

  > 如果指定的数据库 / MinIO BUCKET 不存在，程序会自动创建好，同时确保所有表的定义存在

- `--set-default`：本项目提供了一组默认数据（包含一个默认用户、多个默认教师等已经定义好的初始数据），可以用来用于初始化数据库

  > 必须使用了 `--reset` 参数之后才能生效

2. **访问系统**:
   - 前端页面：`http://localhost:8000`
   - API 文档：`http://localhost:8000/docs`
3. **停止应用**
   - 如果运行在后台模式，可以通过 `kill $(cat logs/service.pid)` 停止服务。
   - 如果运行在前台模式，直接通过 `ctrl+C` 终止运行即可