**《开源技术与应用》**

**大作业设计报告**

**学 院: 计算机学院**

**班 级: 特软2班**

**队 长 姓 名: 屈鑫**

**2025年5月**

**1**概述

本大作业设计并实现了一个基于 FastAPI 的简易会议管理系统。该系统主要提供三大功能：会议信息的管理（创建、查询、更新、删除）、员工信息的管理（创建、查询、更新、删除）、会议预订与查询功能（为员工预订会议、查询员工已预订会议、查询会议的与会人员、取消预订等）。

系统后端采用 Python 和 FastAPI 框架，利用 SQLAlchemy 作为 ORM 与数据库（支持 SQLite 及 MySQL）进行异步交互，Pydantic 用于数据模型的定义与验证。

前端资源通过 static 和 templates 目录进行管理，并使用Jinja2作为模板引擎。项目还具备自动生成 RESTful API 文档（Swagger UI 和 ReDoc）的能力，方便接口的测试与查阅。

**2需求**

本项目满足以下核心需求：

1. 会议管理：对会议进行增删改查操作。
2. 员工管理：对员工信息进行增删改查操作。
3. 会议预订：允许为员工预订特定会议，并能取消预订。
4. 信息查询：能够查询特定员工预订的所有会议，以及特定会议的所有与会人员。
5. API 接口：提供一套完整的 RESTful API 供系统各部分或外部应用调用。
6. 数据持久化：使用数据库存储会议、员工及预订信息。

3系统设计与架构

3.1 技术栈

系统主要采用以下技术栈：

1. 后端框架： FastAPI
2. ORM 框架： SQLAlchemy (支持异步操作 aiosqlite / asyncmy)
3. 数据验证： Pydantic
4. 数据库： SQLite (默认开发) / MySQL (生产环境可选)
5. 模板引擎： Jinja2
6. Web 服务器： Uvicorn

3.2 项目结构

项目采用了模块化的结构，清晰地分离了不同功能的代码，具体如下：

conference/

├── app/

│ ├── \_\_init\_\_.py

│ ├── main.py # 主应用文件

│ ├── models/ # 数据库模型 (SQLAlchemy)

│ │ ├── \_\_init\_\_.py

│ │ ├── database.py # 数据库会话和引擎配置

│ │ ├── conference.py # 会议表模型 (ConferenceDB)

│ │ ├── employee.py # 员工表模型 (EmployeeDB)

│ │ └── booking.py # 预定关联模型

│ ├── schemas/ # Pydantic 数据验证模型

│ │ ├── \_\_init\_\_.py

│ │ ├── conference.py # 会议相关的 Pydantic schema

│ │ ├── employee.py # 员工相关的 Pydantic schema

│ │ └── booking.py # 预定相关的 Pydantic schema

│ ├── api/ # API 路由层

│ │ ├── \_\_init\_\_.py

│ │ ├── conference.py # 会议相关 API 路由

│ │ ├── employee.py # 员工相关 API 路由

│ │ └── booking.py # 预定相关 API 路由

│ └── core/ # 核心配置

│ ├── \_\_init\_\_.py

│ └── config.py # 配置文件读取（如 .env）

├── static/ # 静态文件 (CSS, JavaScript, 图片等)

├── templates/ # HTML 模板文件 (Jinja2)

├── source/ # 其他资源文件

│ └── bg.png # 系统背景图片示例

└── requirements.txt # Python 依赖包列表

3.3 数据库模型设计

系统定义了以下的数据库模型：

1. ConferenceDB: 存储会议的详细信息，如名称、日期、地点、描述。
2. EmployeeDB: 存储员工的基本信息，如姓名、邮箱。
3. EmployeeConferenceDB:用于表示员工和会议之间的多对多关系，记录哪些员工参加了哪些会议。

4实现细节

4.1 后端实现

1. FastAPI 应用：app/main.py 初始化 FastAPI 应用，并包含应用配置，如挂载静态文件目录、模板目录，以及启动/关闭事件处理。
2. API 路由：在app/api/ 目录下，根据资源（会议、员工、预订）分别定义了 API 路由。每个路由模块处理对应资源的 HTTP 请求，并调用相应的服务逻辑。
3. 数据模型：app/models/目录下的 Python 文件使用 SQLAlchemy 定义了与数据库表对应的 ORM 模型。
4. 数据验证：app/schemas/目录下的 Python 文件使用 Pydantic 定义了数据传输对象 (DTOs) 和验证规则，确保 API 输入输出数据的有效性和一致性。
5. 数据库交互： app/models/database.py 配置数据库连接（从 .env 文件读取 DATABASE\_URL），并提供异步数据库会话的依赖注入。所有数据库操作均设计为异步，以提高并发性能。
6. 配置管理： app/core/config.py 负责加载环境变量。

4.2 前端交互

系统具备一个由服务器渲染的 HTML 前端界面。static/ 目录用于存放 CSS、JavaScript 和图片等静态资源，templates/ 目录存放 HTML 模板，由 FastAPI 通过 Jinja2 引擎渲染后返回给客户端浏览器。

4.3 API 端点设计

系统提供了以下 RESTful API 端点：

1. 会议管理:
2. GET /api/conferences: 获取所有会议
3. POST /api/conferences: 创建新会议
4. GET /api/conferences/{id}: 获取指定会议
5. PUT /api/conferences/{id}: 更新会议
6. DELETE /api/conferences/{id}: 删除会议
7. 员工管理:
8. GET /api/employees: 获取所有员工
9. POST /api/employees: 创建新员工
10. GET /api/employees/{id}: 获取指定员工
11. PUT /api/employees/{id}: 更新员工
12. DELETE /api/employees/{id}: 删除员工
13. 会议预订:
14. POST /api/conferences/{id}/book: 预定会议
15. GET /api/employees/{id}/conferences: 获取指定员工预订的所有会议
16. GET /api/conferences/{id}/attendees: 获取指定会议的所有与会人员
17. GET /api/conferences/bookings: 获取所有预定记录
18. DELETE /api/conferences/{id}/bookings/{employee\_id}: 取消指定员工对指定会议的预定

5系统测试与结果

5.1 功能测试

1. API 文档：启动服务后，可以通过访问 http://localhost:8000/docs (Swagger UI) 和 http://localhost:8000/redoc (ReDoc) 查看自动生成的 API 文档。这些文档界面清晰地展示了所有 API 端点、请求参数、请求体结构和响应模型。
2. 会议管理：通过 API 文档界面测试了会议的增删改查功能，均按预期工作。数据能正确存入数据库并被检索和修改。
3. 员工管理：类似地，员工的增删改查功能也通过 API 文档进行了测试，结果符合预期。
4. 会议预订：成功为多个员工预订了会议、能够查询到特定员工预订的所有会议列表、能够查询到特定会议的完整与会人员列表、能成功取消员工的会议预订。
   1. 结果分析

本项目成功实现了需求中给出的核心功能，结构清晰，模块化程度高，易于理解和扩展。FastAPI的特性（自动数据验证、API文档生成）极大地简化了开发的难度，提高了开发效率，保障了项目质量。异步数据库操作，为系统性能打下了良好基础。

6总结与未来工作展望

6.1 总结

本实验构建了一个功能完整、结构清晰的会议管理系统。通过运用FastAPI、SQLAlchemy和Pydantic等现代Python技术栈，实现了会议、员工和预订的管理。系统具备良好可维护性和可扩展性，并提供了API接口和自动文档。

6.2 未来工作展望

1. 用户认证与授权： 集成OAuth2等用户认证机制，实现基于角色的访问控制。
2. 前端界面优化：开发一个交互更友好、功能更丰富的前端（例如使用 Vue.js框架）。
3. 通知系统：增加会议开始提醒、自动发送预订成功/取消邮件等功能。
4. 日历集成：与常用日历服务（如 Google Calendar, Outlook Calendar）集成。
5. 高级查询与筛选：提供更复杂的会议和员工查询筛选功能，包含更多的会议详细信息。
6. 容器化部署： 使用 Docker 和 Docker Compose 进行容器化，简化部署流程。