

南阳理工学院·计算机与软件学院

**《Python程序开发课程设计》报告**

实验管理系统的设计与实现

专 业： 软工智能

班 级： 24软工智能一班

学 号： 2415929709

姓 名： 袁鑫晨

指导老师： 郭东恩

报告日期： 2022年06月xx日

南阳理工学院

Nanyang Institute of Technology

Python程序开发课程设计·成绩评定表

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 考核方式 | 评分项目 | 课程目标 | 总分 | 得分 | 合计 | 占比 |
| 材料考核 | 课程设计报告 | 课程目标1 |  |  | 43 | 50% |
| 课程目标3 |  |  |
| 实践考核 | 课程设计答辩 | 课程目标1 |  |  | 48 | 50% |
| 课程目标2 |  |  |
| 最终成绩： | | |  | | | |
| 指导老师签字： | | |  | | | |

**1.材料考核环节成绩评分标准**

指导教师根据问题分析与定义，逻辑结构设计和详细设计的正确性等综合实训报告情况，按照百分制评定成绩，占总评成绩的50%。评分标准如5所示。

表5 材料考核评分标准

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **课程目标** | **考核指标** | **等级** | | | | |
| **优秀** | **良好** | **中等** | **及格** | **不及格** |
| **课程目标1：**根据给定的复杂工程问题，能够选择合适的函数和模块等，设计高效的程序流程方案并开发实现，分析、解释实验结果，判断程序结果是否符合预期，给出合理有效的结论。 | 函数、模块、技术方案、算法描述、实验结果分析及优化改进思路 | 数学模型正确、技术方案可行、程序流程描述完整；运行结果完全正确，能够对结果的进行准确、全面的分析评价，有明确的优化改进思路。 | 数学模型正确、技术方案可行、程序流程描述比较完整；运行结果正确，能够对结果的进行准确的分析评价，提出了一定的优化改进思路。 | 数学模型比较正确、技术方案可行、程序流程描述比较完整；运行结果正确，能够对结果进行分析评价。 | 数学模型基本正确、技术方案基本可行、程序流程步骤描述基本完整；运行结果基本正确，能够对结果的进行一定程度的分析评价。 | 数学模型、技术方案、程序流程描述等存在多处错误或缺失；运行结果不正确，不能对结果进行正确的分析评价。 |
| **课程目标3：**具备分析和描述软件开发过程的能力，能够系统规范的撰写软件开发设计文档，设计功能流程、简述实现过程、验证实现结果。 | 综合复杂性问题的分析和处理 | 能正确使用测试数据验证程序，测试结果分析全面且正确。 | 能正确使用测试数据验证程序，测试结果分析不够全面。 | 能正确使用测试数据验证程序，测试结果分析部分正确。 | 能使用测试数据验证程序，分析测试结果。 | 不能使用测试数据验证程序，无分析测试结果。 |

表6 实践考核成绩评分标准

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **课程目标** | **考核指标** | **等级** | | | | |
| **优秀** | **良好** | **中等** | **及格** | **不及格** |
| **课程目标1：**根据给定的复杂工程问题，能够选择合适的函数和模块等，设计高效的程序流程方案并开发实现，分析、解释实验结果，判断程序结果是否符合预期，给出合理有效的结论。 | 函数、模块、技术方案、算法描述、实验结果分析及优化改进思路 | 数学模型正确、技术方案可行、程序流程描述完整；运行结果完全正确，能够对结果的进行准确、全面的分析评价，有明确的优化改进思路。 | 数学模型正确、技术方案可行、程序流程描述比较完整；运行结果正确，能够对结果的进行准确的分析评价，提出了一定的优化改进思路。 | 数学模型比较正确、技术方案可行、程序流程描述比较完整；运行结果正确，能够对结果进行分析评价。 | 数学模型基本正确、技术方案基本可行、程序流程步骤描述基本完整；运行结果基本正确，能够对结果的进行一定程度的分析评价。 | 数学模型、技术方案、程序流程描述等存在多处错误或缺失；运行结果不正确，不能对结果进行正确的分析评价。 |
| **课程目标2：**具备依据Python课程设计实际问题的需求，运用所学知识对具体实际项目进行设计的能力。 | 程序质量，程序结构、代码规范性、可读性、测试样例的正确性。 | 程序运行稳定且功能完善。程序结构清晰，代码规范，可读性强。测试数据正确。 | 程序运行稳定且正确，功能全。程序结构清晰，代码规范，可读性强，测试数据正确。 | 程序运行稳定且正确，功能不够完善。代码规范，可读性强，测试数据正确。 | 程序运行稳定且正确。代码不规范，测试数据基本正确。 | 程序运行不稳定，测试数据不确。 |

实验室管理系统的设计与实现

软件工程 袁鑫晨

1. 传统数据库操作中，手动调用cursor.close()和connection.close()容易因代码分支遗漏或异常中断导致资源未释放，长期运行可能引发连接池溢出或内存泄漏。上下文管理器优势：通过@contextmanager装饰器构建的cursor上下文，利用 Python 的with语句强制在代码块结束后执行finally逻辑，确保无论是否发生异常，cursor和底层connection都会被正确关闭。
2. Sqlite,数据库，系统,pyqt, python

Design and Implementation of the  
 Laboratory Management System System

Software Engineering Major Yuan Xinchen

1. In traditional database operations, manually invoking cursor.close() and connection.close() can lead to resource leaks due to overlooked code branches or unexpected exceptions. Prolonged operation under such circumstances may result in connection pool overflows or memory leaks. Context managers, constructed using Python's @contextmanager decorator, enforce resource cleanup through the with statement. This ensures that finally blocks are executed regardless of exceptions, guaranteeing proper closure of cursors and database connections.
2. SQLite, Database, System, PyQt, Python

目 录

[Python程序开发课程设计·成绩评定表 1](#_Toc200148253)

[1 绪论 1](#_Toc200148254)

[1.1 研究背景 1](#_Toc200148255)

[1.2 研究目的与意义 1](#_Toc200148256)

[1.2.1 研究目的 1](#_Toc200148257)

[1.2.2 研究意义 1](#_Toc200148258)

[1.3 国内外研究现状 1](#_Toc200148259)

[1.3.1 国外研究现状 1](#_Toc200148260)

[1.3.2 国内研究现状 1](#_Toc200148261)

[1.4 研究内容 1](#_Toc200148262)

[1.4.1 多角色权限管理 1](#_Toc200148263)

[1.4.2 实验任务全流程管理 1](#_Toc200148264)

[1.4.3 数据持久化存储 1](#_Toc200148265)

[1.4.4 跨平台 GUI 开发 2](#_Toc200148266)

[2 相关技术简介 1](#_Toc200148267)

[2.1 Python 编程语言 1](#_Toc200148268)

[2.2 PyQt5 框架 1](#_Toc200148269)

[2.3 QSS（Qt Style Sheets） 1](#_Toc200148270)

[2.4 SQLite 数据库 2](#_Toc200148271)

[2.5 使用MVC模式 2](#_Toc200148272)

[2.6 数据库连接的自动管理 2](#_Toc200148273)

[2.7 PyInstaller打包独立程序 2](#_Toc200148274)

[3 需求描述与分析 1](#_Toc200148275)

[3.1 功能需求 1](#_Toc200148276)

[3.1.1 用户角色与权限 1](#_Toc200148277)

[3.1.2 核心功能模块 1](#_Toc200148278)

[3.2 非功能需求 2](#_Toc200148279)

[3.2.1 安全性要求​ 2](#_Toc200148280)

[**3.2.2 扩展性要求​** 2](#_Toc200148281)

[4 系统设计 1](#_Toc200148282)

[4.1 总体架构 1](#_Toc200148283)

[4.2 目录结构 1](#_Toc200148284)

[4.3 系统GUI页面 2](#_Toc200148285)

[**4.3.1 登录界面(login\_window.ui)** 2](#_Toc200148286)

[**4.3.2 注册界面(register\_window.ui)** 2](#_Toc200148287)

[**4.3.3 主界面(个人信息main\_window.ui)** 2](#_Toc200148288)

[**4.3.4 主界面(查看实验main\_window.ui)** 3](#_Toc200148289)

[**4.3.5 主界面(实验管理main\_window.ui)** 4](#_Toc200148290)

[**4.3.6 主界面(用户管理main\_window.ui)** 5](#_Toc200148291)

[**4.3.7 编辑/添加用户界面(user\_information.ui)** 6](#_Toc200148292)

[**4.3.8 编辑/添加实验界面(create\_experiment.ui)** 7](#_Toc200148293)

[**4.3.9 批改/提交实验界面(grade\_submission.ui,** **submit\_experiment.ui)** 7](#_Toc200148294)

[4.4 核心界面逻辑 8](#_Toc200148295)

[4.4.1 登录界面(login\_window.py, register\_window.py) 8](#_Toc200148296)

[4.4.2 主窗口界面 (main\_window.py) 8](#_Toc200148297)

[4.4.3 子界面 (create\_experiment.py, grade\_submission.py, submit\_experiment.p，user\_information.pyy) 8](#_Toc200148298)

[4.5 核心工具 8](#_Toc200148299)

[**4.5.1 目录工具** 8](#_Toc200148300)

[**4.5.2 日志工具** 9](#_Toc200148301)

[**4.5.3 数据库工具** 10](#_Toc200148302)

[4.6 数据库设计 10](#_Toc200148303)

[4.6.1 核心表结构​ 10](#_Toc200148304)

[4.6.2 初始化sql语句 11](#_Toc200148305)

[5 测试 1](#_Toc200148306)

[5.1 多账号并发操作测试​ 1](#_Toc200148307)

[5.1.1 测试目的​ 1](#_Toc200148308)

[5.1.2 测试场景​ 1](#_Toc200148309)

[5.1.3 结果 1](#_Toc200148310)

[5.2 教师添加删除学生测试 1](#_Toc200148311)

[5.2.1 测试目的​ 1](#_Toc200148312)

[5.2.2 测试场景​ 1](#_Toc200148313)

[5.2.3 结果 1](#_Toc200148314)

[总结与展望 2](#_Toc200148315)

[参考文献 3](#_Toc200148316)

# 绪论

## 研究背景

随着高校实验教学改革的深入和科研项目的持续增加，实验室资源管理复杂度显著提升。传统的人工管理方式在用户权限分配、实验任务跟踪、数据存储与查询等方面暴露出效率低下、易出错等问题。同时，多角色协同管理（管理员、教师、学生）对系统的权限分离和功能差异化提出了更高要求。在此背景下，开发一套基于 PyQt5 和 SQLite3 的实验室管理系统，实现信息化、规范化管理，成为提升实验室运行效率的迫切需求。

## 研究目的与意义

### 研究目的

设计并实现一个跨平台的实验室管理系统，满足三端用户（管理员、教师、学生）的核心功能需求，包括权限分离的用户管理、全流程实验任务管理、结构化数据存储与高效查询。

### 研究意义

通过信息化手段优化实验室管理流程，降低人工成本；实现数据的集中存储与安全共享，为教学科研提供数据支撑；培养软件开发实践能力，掌握 PyQt5 界面设计与 SQLite3 数据库交互的核心技术。

## 国内外研究现状

### 国外研究现状

发达国家较早实现实验室管理信息化，如美国高校普遍采用集成化管理系统（如 LabVantage），支持设备监控、实验预约、数据追溯等功能，技术成熟但成本较高。

### 国内研究现状

国内高校实验室管理系统多为定制化开发，以 Java 或 C# 为主要技术栈，近年来 Python 凭借简洁性和跨平台优势逐渐被采用。现有系统存在功能同质化、扩展性不足等问题，尤其在轻量级桌面应用领域存在优化空间。

## 研究内容

### 多角色权限管理

实现管理员、教师、学生三端独立界面与功能权限控制。​

### 实验任务全流程管理

支持任务发布、作业提交、成绩评定的闭环管理。​

### 数据持久化存储

基于 SQLite3 设计关系型数据库，实现用户、任务、作业数据的结构化存储。​

### 跨平台 GUI 开发

使用 PyQt5 构建响应式界面，通过 QSS 实现主题化样式设计。

# 相关技术简介

## Python 编程语言

Python 是一种高级、通用且解释型的编程语言，以其简洁易读、功能强大、开源免费等显著优势而闻名。在本系统的开发中，Python 扮演着核心角色，贯穿整个系统的开发过程，用于实现系统的业务逻辑处理和数据交互。

## PyQt5 框架

PyQt5 是 Qt 5 框架的 Python 绑定库，提供 500+GUI 组件，支持跨 Windows/macOS/Linux 平台开发。核心特性包括：​

**信号与槽机制:**解耦界面交互与业务逻辑（如登录按钮点击信号连接数据库查询槽函数）。​

**布局管理：**通过 QVBoxLayout/QHBoxLayout 实现响应式布局，适配不同屏幕尺寸。​

参考教程：[1]

## QSS（Qt Style Sheets）

Qt Style Sheets（QSS）是 Qt 的界面美化语言，语法兼容 CSS，支持元素选择器、伪状态和属性设置。本系统典型应用：

\* {

background-color: #ffffff;

color: #808080;

}

QTextBrowser {

border: none; /\* 隐藏边框 \*/

background-color: transparent; /\* 背景颜色 \*/

color: #808080; /\* 文字颜色 \*/

border-radius: 10px; /\* 圆角 \*/

padding: 5px; /\* 内边距 \*/

}

QPushButton {

background-color: #e9e7ef;

border-radius: 10px;

padding: 5px;

border: none; /\* 移除边框 \*/

outline: none; /\* 移除焦点轮廓 \*/

}

QPushButton:hover {

background-color: #c0ebd7;

}

QPushButton:pressed {

background-color: #88ada6;

}

QPushButton:focus {

border: none; /\* 确保焦点时没有边框 \*/

}

QLineEdit {

background-color: #e9e7ef;

border: none; /\* 去掉边框 \*/

border-radius: 10px; /\* 圆角 \*/

padding: 5px;

color: #3d3b4f; /\* 输入字体颜色 \*/

border-bottom: 2px solid #3d3b4f; /\* 底部下划线 \*/

}

QLineEdit:hover {

background-color: #d0d7e5; /\* 鼠标悬停时的背景颜色 \*/

}

QLineEdit:focus {

background-color: #b8c0d0; /\* 获得焦点时的背景颜色 \*/

border-bottom: 2px solid #88ada6; /\* 获得焦点时的下划线颜色 \*/

}

参考教程：[5]

## SQLite 数据库

轻量级嵌入式关系型数据库，具备以下优势：

**单文件存储**：数据存储于单个.db文件，无需服务器配置，适合中小规模数据管理。

**ACID 特性**：支持事务处理，确保数据一致性（如用户删除操作同时更新关联表记录）。

**Python 原生支持**：通过sqlite3模块实现无缝交互，支持参数化查询防止 SQL 注入。

参考教程：[4]

## 使用MVC模式

系统采用 MVC设计核心组件：​

模型（Model）：负责处理应用程序的数据和业务逻辑。

视图（View）：负责显示数据给用户。

控制器（Controller）：负责接收用户的输入并调用模型和视图去完成相应的任务。

​参考教程：[1]

## 数据库连接的自动管理

传统数据库操作中，手动调用cursor.close()和connection.close()容易因代码分支遗漏或异常中断导致资源未释放，长期运行可能引发连接池溢出或内存泄漏。**上下文管理器优势**：通过@contextmanager装饰器构建的cursor上下文，利用 Python 的with语句强制在代码块结束后执行finally逻辑，确保无论是否发生异常，cursor和底层connection都会被正确关闭。

## PyInstaller打包独立程序

该系统使用PyInstaller打包,可独自运行，不需要python程序运行

# 需求描述与分析

## 功能需求

### 用户角色与权限

|  |  |  |
| --- | --- | --- |
| 角色 | 核心功能 | 权限说明 |
| **管理员** | 用户信息管理（增删改查）  ,查看所有实验,管理用户权限 | 可增删改查所有用户基础信息（用户名、真实姓名、学号 ）,实时监控所有实验任务进度及提交状态,修改用户密码） |
| **教师** | 实验创建与编辑，实验提交，批改 | 仅能操作自己创建的实验任务,管理绑定的学生,可打回未评分提交，重新批改已评分实验（记录批改历史）生成多维度成绩报表（按学生 / 实验 / 班级统计平均分、优秀率） |
| **学生** | 实验查看与提交实验报告，查询个人信息 | 仅能查看教师分配的实验任务（按课程 / 截止时间排序）未评分提交可无限次修改，已评分提交支持查看但不可编辑实时查询成绩及教师评语，支持成绩申诉（需管理员审核）维护个人基础信息（联系方式、实验环境偏好），不可查看他人信息 |

### 核心功能模块

**登录登出模块（**[login\_window.py](http://user_information.py/)，[register\_window.py](http://user_information.py/" \t "_blank)**）**

**用户管理模块** ([user\_information.py](http://user_information.py/))​

基础功能：支持用户信息多条件查询（姓名模糊搜索）

权限控制：管理员可以更改用户权限

**实验提交模块** ([submit\_experiment.py](http://submit_experiment.py/))​

基础功能：支持教师批改，打回.支持学生查看自己的实验分数，和评语

搜索功能：利用数据库的查询语句,模糊查询实验

**实验批改模块** (grade\_submission.ui)​

批改管理：标记提交状态（待批改 / 已批改 ）

交互功能：内置评语输入

## 非功能需求

### 安全性要求​

**数据加密**：用户密码采用 SHA-256 算法加密

**注入防护**：所有数据库操作使用参数化查询（禁止字符串拼接）

**审计跟踪**：记录用户登录日志(登录时间）敏感操作日志（操作类型、影响数据、执行时间）

### **扩展性要求​**

**工具复用**：提供通用工具类（数据库连接工具[db\_utils.py](http://db_utils.py/)、加密工具[hash\_utils.py](http://hash_utils.py/)、日志工具[log\_utils.py](http://log_utils.py/)），支持跨模块调用​

# 系统设计

## 总体架构

## 目录结构

实验室管理系统/ # 项目根目录

├── main.py # 程序入口

│

├── database/ # 数据库目录

│ ├── database.db # SQLite数据库文件

│ ├── migrations/ # 数据库迁移脚本

│ ├── models/ # 用户数据模型

│ └── \_\_init\_\_.py # 初始化文件(14行)

│

├── logs/ # 日志目录

│ ├── lab\_manager\_20250606.log # 最新日志

│ ├── lab\_manager\_20250605.log # 历史日志

│ ├── lab\_manager\_20250604.log # 历史日志

│ └── lab\_manager\_20250603.log # 历史日志

│

├── resources/ # 资源目录

│ └── icons/ # 图标目录

│ ├── app\_icon.ico # 应用图标(4.2KB)

│ └── app\_icon.svg # SVG图标(751B)

│

├── utils/ # 工具类目录

│ ├── db\_utils.py # 数据库工具

│ ├── hash\_utils.py # 加密工具

│ ├── logger.py # 日志工具

│ ├── validation\_utils.py # 验证工具

│ ├── path\_helper.py # 路径工具

│ ├── icon\_helper.py # 图标工具

│ ├── file\_handler.py # 文件处理

│ └── \_\_init\_\_.py # 初始化文件

│

└── ui/ # 界面目录

├── login/ # 登录界面

├── menu/ # 主菜单界面

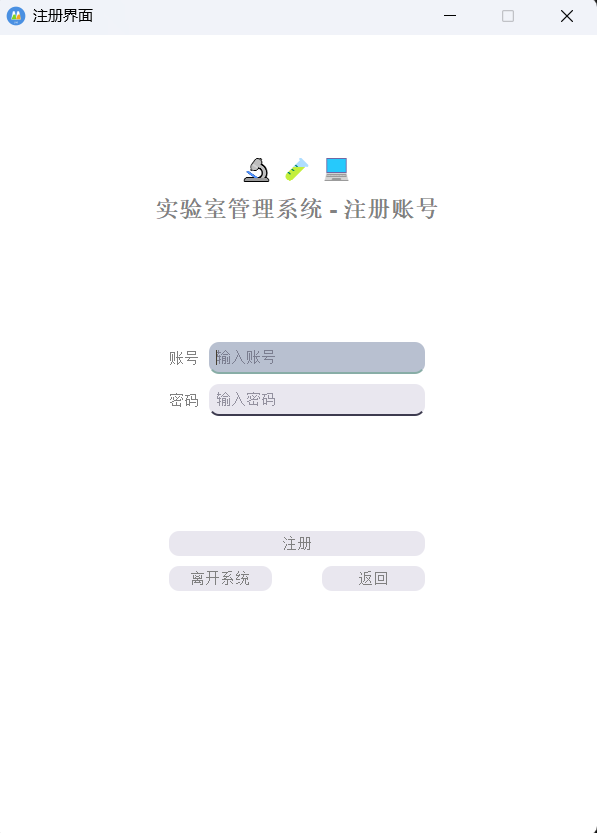
└── ui\_tools.py # UI工具

## 系统GUI页面

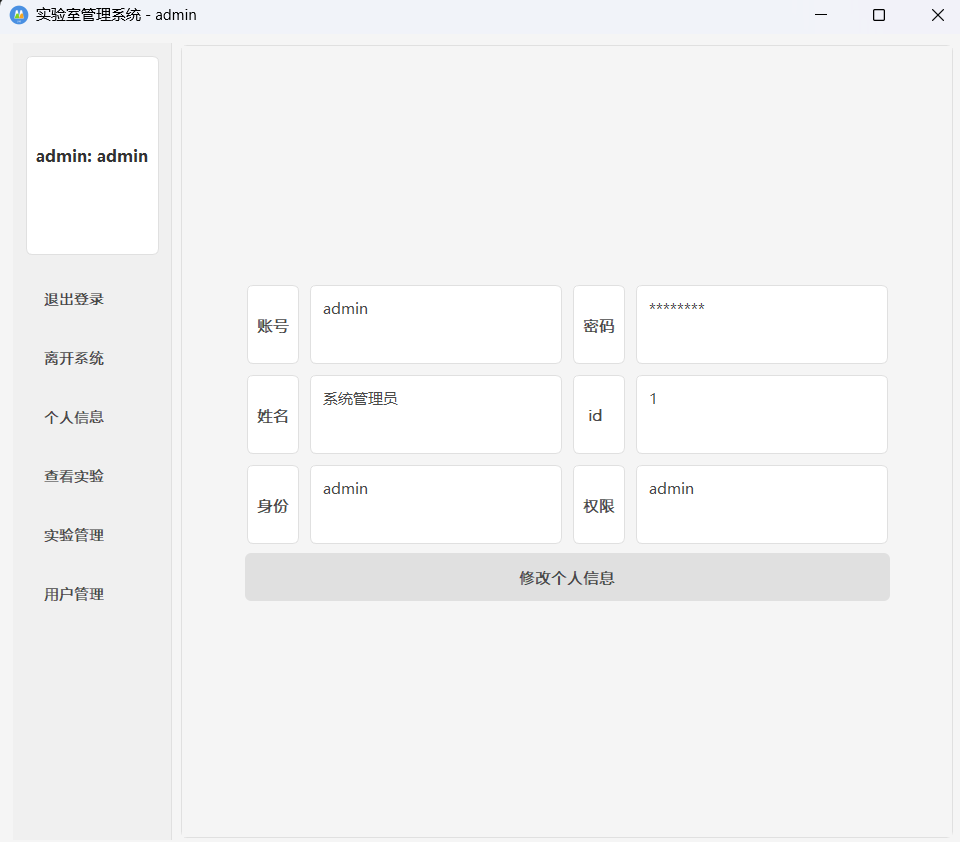
### **登录界面(login\_window.ui)**



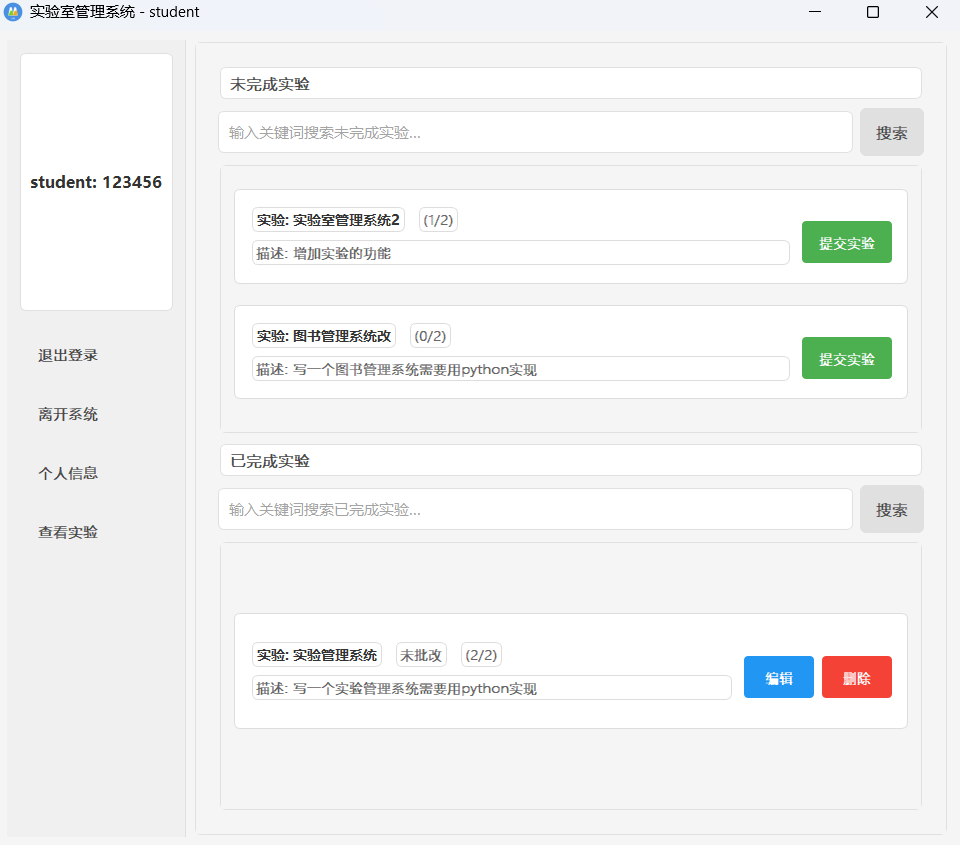
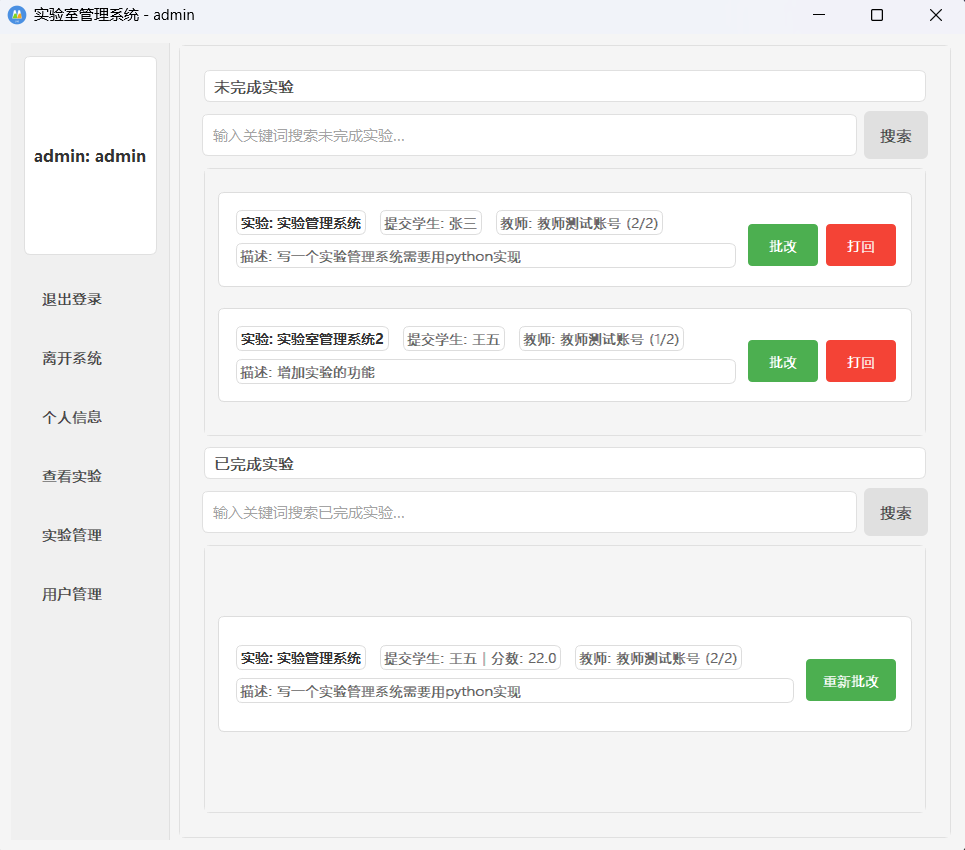
### **注册界面(register\_window.ui)**



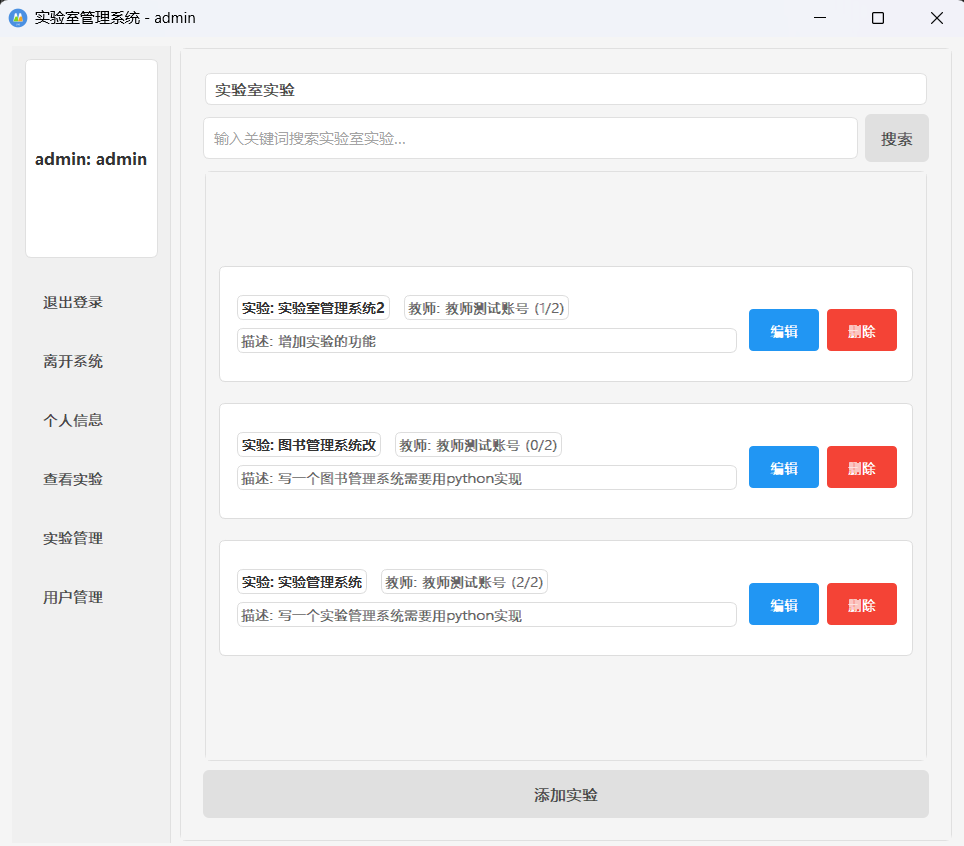
### **主界面(个人信息main\_window.ui)**



### **主界面(查看实验main\_window.ui)**



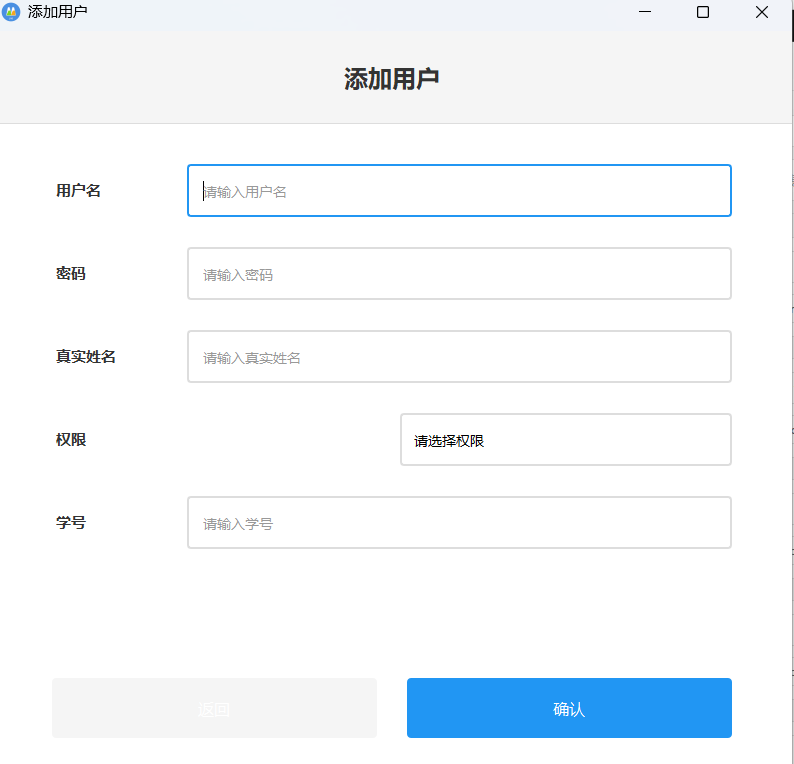
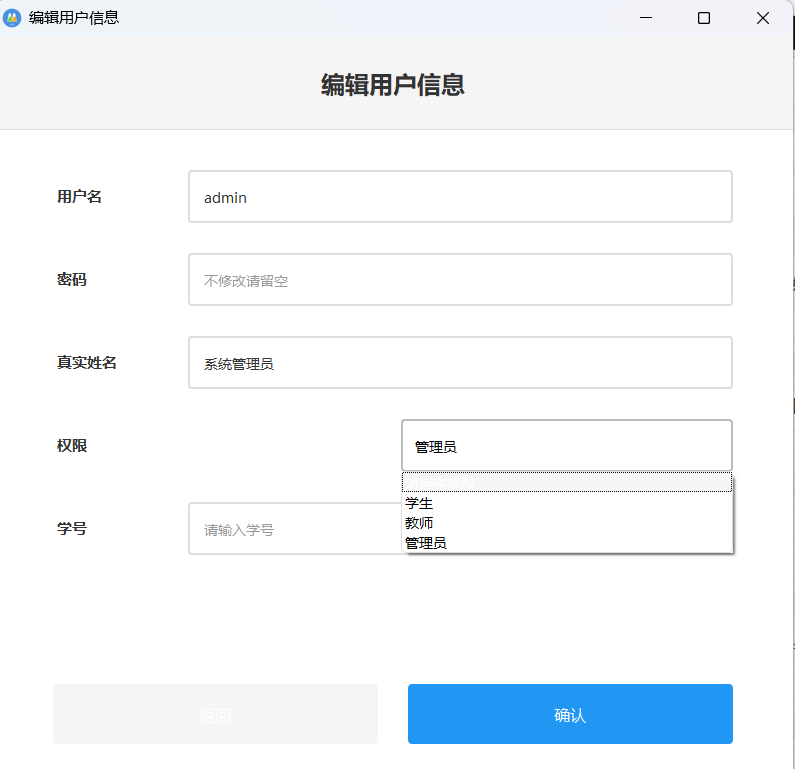
### **主界面(实验管理main\_window.ui)**



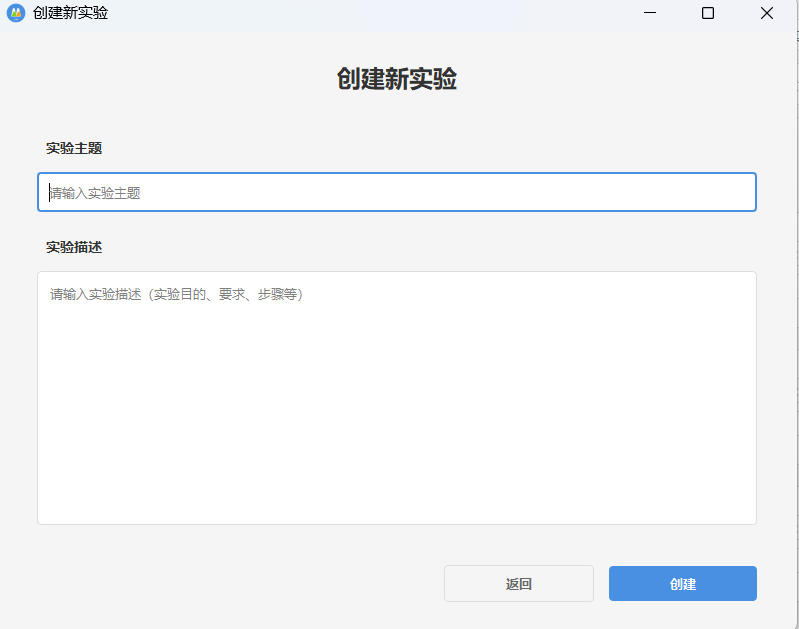
### **主界面(用户管理main\_window.ui)**



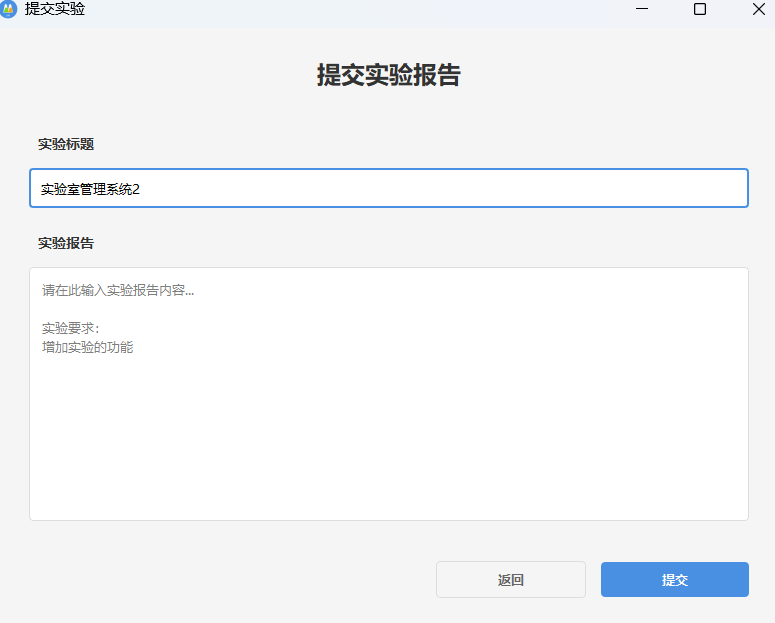
### **编辑/添加用户界面(user\_information.ui)**



### **编辑/添加实验界面(create\_experiment.ui)**



### **批改/提交实验界面(grade\_submission.ui,** **submit\_experiment.ui)**



## 核心界面逻辑

### 登录界面(login\_window.py, register\_window.py)

**功能职责**：作为界面入口的实现逻辑，在其内部实现登录和注册等操作

**主要类**：LoginWindow(QWidget) RegisterWindow(QWidget)

**核心方法**：Login

**功能:** 实现登录操作

def **login**(*self*):

        """登录功能"""

        info(f"用户尝试登录")

        username = *self*.lineEdit\_7.text().strip()  *# 获取账号*

        password = *self*.lineEdit\_5.text().strip()  *# 获取密码*

*self*.clear\_error()

*# 验证输入不能为空*

*if* not username or not password:

            warning("用户名或密码为空")

*self*.show\_error("用户名和密码不能为空！")

*return*

*try*:

*# 验证用户名格式*

            is\_valid, error\_msg = vu.validate\_username(username)

*if* not is\_valid:

                warning(f"用户名格式错误: {error\_msg}")

*self*.show\_error(error\_msg)

*return*

*# 验证密码格式*

            is\_valid, error\_msg = vu.validate\_password(password)

*if* not is\_valid:

                warning(f"密码格式错误: {error\_msg}")

*self*.show\_error(error\_msg)

*return*

*#对密码进行哈希加密*

            password = hu.hash\_string(password)

*# 创建用户对象并尝试登录*

            user = User()

*if* user.login(username, password):

*self*.current\_user = user

                info(f"用户 {username} 登录成功")

*self*.login\_success()

*else*:

                warning(f"用户 {username} 登录失败")

*self*.show\_error("用户名或密码错误！")

*except* ValueError *as* e:

            warning(f"登录验证错误: {str(e)}")

*self*.show\_error("用户名或密码错误！")

*except* Exception *as* e:

            error(f"登录过程发生错误: {str(e)}")

            exception("登录异常详细信息")

            QMessageBox.critical(*self*, "错误", "登录失败，请稍后重试！")

### 主窗口界面 ([main\_window.py](http://main_window.py/))

**功能职责**：作为主要窗口，在其内部实现操作界面逻辑，动态加载实验

**主要类**：MainWindow(QMainWindow)​

**核心方法**：display\_experiments

**功能:** 在图形界面（GUI）中渲染并展示实验列表适用于学生、教师或管理员等不同角色的用户，根据角色和实验状态（是否完成 / 批改）动态显示对应的实验信息和操作按钮。

def display\_experiments(self, experiments, layout, is\_finished: bool = False):

"""显示实验列表

Args:

experiments: 实验信息列表（每个元素为实验数据元组，结构依赖业务逻辑）

layout: 要显示到的目标布局（QVBoxLayout/QHBoxLayout等容器）

is\_finished: 是否显示已完成/已批改实验（控制按钮和状态显示）

"""

# ---------------------- 初始化布局 ----------------------

# 清空现有内容，避免重复渲染

self.clear\_layout(layout)

# 添加顶部间距（垂直方向固定间隔）

spacer\_top = QtWidgets.QSpacerItem(

20, 10, # 宽度20，高度10

QtWidgets.QSizePolicy.Minimum,

QtWidgets.QSizePolicy.Fixed

)

layout.addItem(spacer\_top)

# ---------------------- 遍历渲染每个实验项 ----------------------

for exp in experiments:

# ---------------------- 创建实验项容器 ----------------------

frame = QtWidgets.QFrame()

frame.setFrameShape(QtWidgets.QFrame.Box) # 设置边框样式为方框

frame.setSizePolicy(

QtWidgets.QSizePolicy.Expanding, # 水平方向可扩展

QtWidgets.QSizePolicy.Fixed # 垂直方向固定高度

)

frame.setMinimumHeight(90) # 最小高度（防止内容挤压）

frame.setMaximumHeight(120) # 最大高度（限制内容过多时的膨胀）

frame.setStyleSheet("""

QFrame {

background-color: white; /\* 白色背景 \*/

border: 1px solid #dddddd; /\* 浅灰色边框 \*/

border-radius: 6px; /\* 圆角边框 \*/

}

""")

# ---------------------- 实验项内部布局 ----------------------

v\_layout = QtWidgets.QVBoxLayout(frame) # 垂直主布局

v\_layout.setContentsMargins(15, 12, 15, 12) # 内边距（上下左右）

v\_layout.setSpacing(6) # 子组件垂直间距

# ---------------------- 标题行和按钮行的水平布局 ----------------------

h\_layout = QtWidgets.QHBoxLayout() # 水平布局（左信息 + 右按钮）

h\_layout.setSpacing(10) # 左右部分间距

# ---------------------- 左侧信息区域 ----------------------

info\_layout = QtWidgets.QVBoxLayout() # 信息垂直布局

info\_layout.setSpacing(4) # 信息行间距

# ---------------------- 第一行：标题、状态、统计信息 ----------------------

title\_layout = QtWidgets.QHBoxLayout() # 标题行水平布局

title\_layout.setSpacing(10) # 标题与状态间距

# 实验标题（限制长度，超出显示省略号）

title = exp[2]

if len(title) > 30:

title = title[:30] + "..."

title\_label = QtWidgets.QLabel(f"实验: {title}")

title\_label.setStyleSheet("""

font-size: 14px; /\* 字体大小 \*/

font-weight: bold; /\* 加粗 \*/

color: #333333; /\* 深灰色文字 \*/

padding: 2px 0; /\* 上下内边距 \*/

""")

title\_layout.addWidget(title\_label)

# ---------------------- 不同角色显示不同状态信息 ----------------------

if self.role in ['teacher', 'admin']:

# 教师/管理员视图：显示学生姓名和分数

if exp[8]: # student\_name存在

student\_info = f"提交学生: {exp[8]}"

if exp[6] is not None: # 有分数时显示

student\_info += f" | 分数: {exp[6]}"

student\_label = QtWidgets.QLabel(student\_info)

student\_label.setStyleSheet("""

color: #666666; /\* 中灰色文字 \*/

font-size: 13px;

padding: 2px 0;

""")

title\_layout.addWidget(student\_label)

elif self.role == 'student' and is\_finished:

# 学生视图（已完成实验）：显示分数或状态

score\_info = f"分数: {exp[6]}" if exp[6] is not None else "未批改"

score\_label = QtWidgets.QLabel(score\_info)

score\_label.setStyleSheet("""

color: #666666;

font-size: 13px;

padding: 2px 0;

""")

title\_layout.addWidget(score\_label)

# 提交统计信息（提交人数/总人数）

if len(exp) > 8: # 确保数据存在

submit\_count = exp[-2]

total\_students = exp[-1]

progress = f"({submit\_count}/{total\_students})"

if self.role == 'admin':

# 管理员额外显示教师姓名

progress = f"教师: {exp[9]} " + progress

status\_label = QtWidgets.QLabel(progress)

status\_label.setStyleSheet("""

color: #666666;

font-size: 13px;

padding: 2px 0;

""")

title\_layout.addWidget(status\_label)

title\_layout.addStretch() # 右侧空白填充，使按钮右对齐

info\_layout.addLayout(title\_layout) # 添加到信息布局

# ---------------------- 第二行：实验描述 ----------------------

if exp[3]: # 存在描述时显示

desc = exp[3]

if len(desc) > 50: # 限制描述长度

desc = desc[:50] + "..."

desc\_label = QtWidgets.QLabel(f"描述: {desc}")

desc\_label.setStyleSheet("""

color: #666666;

font-size: 13px;

padding: 2px 0;

""")

info\_layout.addWidget(desc\_label)

h\_layout.addLayout(info\_layout, stretch=1) # 左侧信息占主要拉伸空间

# ---------------------- 右侧按钮区域 ----------------------

button\_layout = QtWidgets.QHBoxLayout() # 按钮水平布局

button\_layout.setSpacing(8) # 按钮间距

# ---------------------- 根据用户角色动态显示按钮 ----------------------

if self.role == 'student':

if not is\_finished:

# 学生未完成实验：显示提交按钮

submit\_btn = QtWidgets.QPushButton("提交实验")

submit\_btn.setFixedSize(90, 30) # 固定宽高

submit\_btn.setStyleSheet("""

QPushButton {

background-color: #4CAF50; /\* 绿色按钮 \*/

color: white;

border-radius: 4px;

font-size: 13px;

}

QPushButton:hover { background-color: #45a049; } /\* 悬停加深颜色 \*/

""")

# 绑定点击事件，传递实验ID

submit\_btn.clicked.connect(lambda \_, eid=exp[0]: self.submit\_experiment(eid))

button\_layout.addWidget(submit\_btn)

else:

# 学生已完成实验：根据评分状态显示编辑/删除/查看按钮

if exp[6] is None: # 未评分时允许编辑和删除

edit\_btn = QtWidgets.QPushButton("编辑", fixedSize=(70, 30))

edit\_btn.setStyleSheet("""

background-color: #2196F3; /\* 蓝色按钮 \*/

color: white;

border-radius: 4px;

""")

edit\_btn.clicked.connect(lambda \_, eid=exp[0]: self.edit\_submission(eid))

button\_layout.addWidget(edit\_btn)

delete\_btn = QtWidgets.QPushButton("删除", fixedSize=(70, 30))

delete\_btn.setStyleSheet("""

background-color: #F44336; /\* 红色按钮 \*/

color: white;

border-radius: 4px;

""")

delete\_btn.clicked.connect(lambda \_, eid=exp[0]: self.delete\_submission(eid))

button\_layout.addWidget(delete\_btn)

else: # 已评分时仅允许查看

view\_btn = QtWidgets.QPushButton("查看", fixedSize=(70, 30))

view\_btn.setStyleSheet("""

background-color: #757575; /\* 灰色按钮 \*/

color: white;

border-radius: 4px;

""")

view\_btn.clicked.connect(lambda \_, eid=exp[0]: self.view\_submission(eid))

button\_layout.addWidget(view\_btn)

else:

# 教师/管理员视图：根据页面类型显示管理按钮

if layout == self.verticalLayout\_11: # 实验管理页面（列表页）

edit\_btn = QtWidgets.QPushButton("编辑", fixedSize=(70, 30))

edit\_btn.setStyleSheet("""background-color: #2196F3; color: white; border-radius: 4px;""")

edit\_btn.clicked.connect(lambda \_, eid=exp[0]: self.edit\_experiment(eid))

button\_layout.addWidget(edit\_btn)

delete\_btn = QtWidgets.QPushButton("删除", fixedSize=(70, 30))

delete\_btn.setStyleSheet("""background-color: #F44336; color: white; border-radius: 4px;""")

delete\_btn.clicked.connect(lambda \_, eid=exp[0]: self.delete\_experiment(eid))

button\_layout.addWidget(delete\_btn)

else:

# 批改实验页面：显示批改/打回/重新批改按钮

if not is\_finished: # 未批改状态

grade\_btn = QtWidgets.QPushButton("批改", fixedSize=(70, 30))

grade\_btn.setStyleSheet("""background-color: #4CAF50; color: white; border-radius: 4px;""")

grade\_btn.clicked.connect(lambda \_, eid=exp[0]: self.grade\_submission(eid))

button\_layout.addWidget(grade\_btn)

return\_btn = QtWidgets.QPushButton("打回", fixedSize=(70, 30))

return\_btn.setStyleSheet("""background-color: #F44336; color: white; border-radius: 4px;""")

return\_btn.clicked.connect(lambda \_, eid=exp[0]: self.return\_submission(eid))

button\_layout.addWidget(return\_btn)

else: # 已批改状态

regrade\_btn = QtWidgets.QPushButton("重新批改", fixedSize=(90, 30))

regrade\_btn.setStyleSheet("""background-color: #4CAF50; color: white; border-radius: 4px;""")

regrade\_btn.clicked.connect(lambda \_, eid=exp[0]: self.grade\_submission(eid))

button\_layout.addWidget(regrade\_btn)

h\_layout.addLayout(button\_layout) # 添加按钮到水平布局

v\_layout.addLayout(h\_layout) # 添加水平布局到主垂直布局

# ---------------------- 将实验项添加到目标布局 ----------------------

layout.addWidget(frame)

# 添加实验项之间的间隔

spacer = QtWidgets.QSpacerItem(20, 10, QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Fixed)

layout.addItem(spacer)

# ---------------------- 添加底部间距 ----------------------

spacer\_bottom = QtWidgets.QSpacerItem(20, 10, QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Fixed)

layout.addItem(spacer\_bottom)

. **核心流程**

先调用 self.clear\_layout(layout) 清除目标布局中的现有组件，避免重复渲染。遍历渲染每个实验项,创建卡片容器 QFrame，设置尺寸和样式。填充左侧信息区域（标题、状态、描述）和右侧按钮区域。将卡片添加到目标布局，并在实验项之间添加间隔。添加底部间距，使界面底部留白更整洁。

### 子界面 (create\_experiment.py, grade\_submission.py, submit\_experiment.p，user\_information.pyy)

**功能职责**：主要实现实验和用户的增删改查逻辑

**主要类**：MainWindow(QMainWindow)​

**核心方法**： save\_experiment

**功能:** 将用户创建的实验保存入数据库中

def save\_experiment(*self*):

        """保存实验"""

*# 获取输入*

        title = *self*.lineEdit.text().strip()

        description = *self*.textEdit.toPlainText().strip()

*# 验证输入*

*if* not title:

            QMessageBox.warning(*self*, "错误", "请输入实验主题")

*return*

*if* not description:

            QMessageBox.warning(*self*, "错误", "请输入实验描述")

*return*

*try*:

*with* db.get\_connection() *as* conn:

*if* *self*.is\_edit:

*# 更新实验*

*if* *self*.user\_info['role'] == 'admin':

*# 管理员可以编辑任何实验*

                        cursor = conn.execute("""

                            UPDATE experiments

                            SET title = ?, description = ?

                            WHERE id = ?

                        """, (title, description, *self*.experiment\_id))

*else*:

*# 其他角色只能编辑自己的实验*

                        cursor = conn.execute("""

                            UPDATE experiments

                            SET title = ?, description = ?

                            WHERE id = ? AND creator\_id = ?

                        """, (title, description, *self*.experiment\_id, *self*.user\_info['id']))

*if* cursor.rowcount == 0:

                        QMessageBox.warning(*self*, "错误", "未找到实验或无权限编辑")

*return*

                    info(f"用户:{*self*.user\_info['username']} 更新实验 {*self*.experiment\_id}")

                    QMessageBox.information(*self*, "成功", "实验更新成功")

*else*:

*# 创建新实验*

                    cursor = conn.execute("""

                        INSERT INTO experiments (title, description, creator\_id, create\_time)

                        VALUES (?, ?, ?, datetime('now', 'localtime'))

                    """, (title, description, *self*.user\_info['id']))

                    info(f"用户:{*self*.user\_info['username']} 创建新实验 {cursor.lastrowid}")

                    QMessageBox.information(*self*, "成功", "实验创建成功")

                conn.commit()

*# 刷新父窗口的实验列表*

*if* *self*.parent and hasattr(*self*.parent, 'load\_all\_experiments'):

*self*.parent.load\_all\_experiments()

*# 关闭窗口*

*self*.close()

*except* Exception *as* e:

            error(f"{'更新' *if* *self*.is\_edit *else* '创建'}实验失败: {str(e)}")

            exception(f"{'更新' *if* *self*.is\_edit *else* '创建'}实验失败: {str(e)}")

            QMessageBox.warning(*self*, "错误", f"{'更新' *if* *self*.is\_edit *else* '创建'}实验失败")

## 核心工具

### **目录工具**

**功能职责**：可以找到一些文件的绝对路径

**核心方法**：**get\_root\_path**()

**功能:** 找到程序运行的根路径

def get\_root\_path():

    """

    获取应用程序根目录

    在开发环境和打包环境下都能正确工作

    """

*if* getattr(sys, 'frozen', False):*#打包环境下 编译器冻结*

*# 打包环境下的路径*

*return* os.path.dirname(sys.executable)*#sys.executable 是当前程序的绝对路径*

*else*:

*# 开发环境下的路径*

*return* os.path.dirname(os.path.dirname(os.path.abspath(\_\_file\_\_)))*#\_\_file\_\_ 是当前文件的绝对路径*

### **日志工具**

**功能职责**：在终端中显示调试记录 ,便于调试

**核心方法**：**get\_root\_path**()

**核心类: Logger:** **oloredFormatter(logging.Formatter):**

**核心方法：\_\_init\_\_(*self*):**

**功能:** 初始化日志管理器

def \_\_init\_\_(self):

# 单例模式检查：若已初始化，直接返回，避免重复创建日志处理器

if Logger.\_initialized:

return

Logger.\_initialized = True # 标记为已初始化，确保后续实例化跳过初始化逻辑

# ---------------------- 初始化日志记录器 ----------------------

# 获取名为'LabManager'的日志记录器（logging模块支持通过名称区分不同日志源）

self.logger = logging.getLogger('LabManager')

self.logger.setLevel(logging.DEBUG) # 设置全局日志级别：接收DEBUG及以上级别日志

# ---------------------- 配置日志格式器 ----------------------

# 控制台格式器：带颜色标识（假设ColoredFormatter为自定义带颜色的格式器）

console\_formatter = ColoredFormatter(

fmt='%(asctime)s [%(levelname)s] %(message)s', # 日志格式：时间 [级别] 内容

datefmt='%Y-%m-%d %H:%M:%S' # 时间格式：年月日 时分秒

)

# 文件格式器：纯文本格式（文件中颜色无效）

file\_formatter = logging.Formatter(

fmt='%(asctime)s [%(levelname)s] %(message)s',

datefmt='%Y-%m-%d %H:%M:%S'

)

# ---------------------- 配置控制台日志输出 ----------------------

console\_handler = logging.StreamHandler(sys.stdout) # 输出到控制台（标准输出流）

console\_handler.setFormatter(console\_formatter) # 应用带颜色的格式器

console\_handler.setLevel(logging.DEBUG) # 控制台处理器级别：显示DEBUG及以上日志

# ---------------------- 配置文件日志输出 ----------------------

# 计算日志目录路径：当前文件的上上级目录（通常为项目根目录）下的logs文件夹

log\_dir = os.path.join(os.path.dirname(os.path.dirname(\_\_file\_\_)), 'logs')

os.makedirs(log\_dir, exist\_ok=True) # 创建目录，若已存在则忽略

# 按日期生成日志文件名（每天一个新文件）

log\_file = os.path.join(

log\_dir,

f'lab\_manager\_{datetime.now().strftime("%Y%m%d")}.log' # 格式：lab\_manager\_20231001.log

)

# 旋转文件处理器：自动分割大文件并保留历史备份

file\_handler = RotatingFileHandler(

filename=log\_file, # 日志文件名

maxBytes=10\*1024\*1024, # 单个文件最大容量：10MB

backupCount=5, # 最多保留5个历史备份文件

encoding='utf-8' # 使用UTF-8编码，支持中文等特殊字符

)

file\_handler.setFormatter(file\_formatter) # 应用纯文本格式器

file\_handler.setLevel(logging.DEBUG) # 文件处理器级别：记录DEBUG及以上日志

# ---------------------- 绑定处理器到日志记录器 ----------------------

self.logger.addHandler(console\_handler) # 绑定控制台处理器

self.logger.addHandler(file\_handler) # 绑定文件处理器

### **数据库工具**

**功能职责**：方便与数据库的连接

**核心方法**：**get\_root\_path**()

**核心类: Logger:** **oloredFormatter(logging.Formatter):**

**核心方法： get\_connection(*self*)**

**功能:** 使用python的上下文管理器，方便数据库的连接

    @contextmanager

    def **get\_connection**(*self*) -> sqlite3.Connection:

        """

        获取数据库连接的上下文管理器

        使用方法:

        with db.get\_connection() as conn:

            cursor = conn.cursor()

            cursor.execute(sql, params)

        Returns:

            sqlite3.Connection: 数据库连接对象

        """

        conn = None

*try*:

            conn = sqlite3.connect(*self*.db\_path)

            debug("数据库连接已建立")

*yield* conn

*except* Exception *as* e:

            error(f"数据库连接错误: {str(e)}")

            exception(f"数据库连接错误: {str(e)}")

*raise* e

*finally*:

*if* conn:

                conn.close()

                debug("数据库连接已关闭")

## 数据库设计

### 核心表结构​

根据系统功能需求，设计 3 张核心数据表，通过外键关联实现数据联动，确保数据完整性与一致性：​

​

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **表名**​ | **主键**​ | **外键关系**​ | **核心字段（非主键）**​ | **索引设计**​ |
| users​ | id (INTEGER, 自增)​ | teacher\_id → users.id（自关联，学生的指导教师）​ | username (UNIQUE, 用户名)password (密码)role (角色: admin/teacher/student)real\_name (真实姓名)student\_id (学号，学生专属)teacher\_id (指导教师 ID，教师 / 管理员为自身 ID)​ | idx\_username (用户名唯一索引)idx\_teacher\_id (指导关系查询索引)​ |
| experiments​ | id (INTEGER, 自增)​ | creator\_id → users.id（实验创建者，关联教师）​ | creator\_id (NOT NULL, 教师 ID)title (NOT NULL, 实验标题)description (实验描述)deadline (截止时间)​ | idx\_creator\_id (按创建者查询索引)idx\_deadline (截止时间范围查询索引)​ |
| submissions​ | id (INTEGER, 自增)​ | student\_id → users.idexperiment\_id → experiments.id​ | student\_id (NOT NULL, 学生 ID)experiment\_id (NOT NULL, 实验 ID)submit\_time (NOT NULL, 提交时间)content (NOT NULL, 实验内容)score (分数，可为空)comment (评语，可为空)​ | idx\_student\_experiment (学生 - 实验联合索引，优化提交记录查询)idx\_submit\_time (提交时间排序索引)​ |

​

### 初始化sql语句

-- 初始化数据库结构

-- 创建用户表

-- 如果，不存在users 就创建表

CREATE TABLE IF NOT EXISTS users (

    id INTEGER PRIMARY KEY AUTOINCREMENT,  -- 用户ID，主键，自动递增 设置主键

    username TEXT NOT NULL UNIQUE,         -- 用户名，不能为空，必须唯一

    password TEXT NOT NULL,                -- 密码，不能为空

    role TEXT NOT NULL,                    -- 用户角色（如admin/teacher/student）

    real\_name TEXT,                        -- 真实姓名，可以为空

    student\_id TEXT,                       -- 学号，可以为空

    teacher\_id INTEGER,                    -- 指导教师ID，对于学生指向其指导教师，对于教师和管理员指向自己

    create\_time TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,  -- 创建时间，默认为当前时间

    FOREIGN KEY (teacher\_id) REFERENCES users(id)     -- teacher\_id 关联到用户表的id

);

-- 创建实验任务表

-- 如果，不存在experiments 就创建表

CREATE TABLE IF NOT EXISTS experiments (

    id INTEGER PRIMARY KEY AUTOINCREMENT,  -- 实验ID，主键，自动递增

    creator\_id INTEGER NOT NULL,           -- 创建者ID（教师），外键关联users表

    title TEXT NOT NULL,                   -- 实验标题，不能为空

    description TEXT,                      -- 实验描述，可以为空

    deadline DATETIME,                     -- 截止时间

    create\_time TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,  -- 创建时间，默认为当前时间

    FOREIGN KEY (creator\_id) REFERENCES users(id)     -- 关联到用户表（创建者）

);

-- 创建作业提交表

-- 如果，不存在submissions 就创建表

CREATE TABLE IF NOT EXISTS submissions (

    id INTEGER PRIMARY KEY AUTOINCREMENT,  -- 提交ID，主键，自动递增

    student\_id INTEGER NOT NULL,           -- 学生ID，外键关联users表

    experiment\_id INTEGER NOT NULL,        -- 实验ID，外键关联experiments表

    submit\_time DATETIME NOT NULL,         -- 提交时间

    content TEXT NOT NULL,               -- 提交的内容

    score FLOAT,                           -- 分数

    comment TEXT,                          -- 评语

    create\_time TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,  -- 创建时间，默认为当前时间

    FOREIGN KEY (student\_id) REFERENCES users(id),           -- 关联到用户表

    FOREIGN KEY (experiment\_id) REFERENCES experiments(id)   -- 关联到实验表

);

-- 在此处添加其他表结构

# 测试

## 多账号并发操作测试​

### 测试目的​

验证多用户并发创建 / 删除实验时的数据一致性，以及跨账号数据同步的实时性​

### 测试场景​

**客户端 A（管理员）**：监控全局数据变化​

**客户端 B（教师 A）**：并发创建 几个实验任务​

**客户端 D（学生）**：实时查看分配的实验任务是否正确更新

### 结果

教师A 添加实验后， 学生刷新界面出现实验

教师A删除实验后,学生刷新界面后实验消失

## 教师添加删除学生测试

### 测试目的​

检测学生制定教师的程序是否正常

### 测试场景​

**客户端 A（管理员）**：监控全局数据变化​

**客户端 B（教师 A）**：移除学生，和添加为自己的学生

**客户端 D（学生）**：实时查看分配的实验任务是否正确更新

### 结果

教师A移除学生后， 学生刷新界面后，分配的实验消失

教师A添加学生后， 学生刷新界面，分配的实验出现

总结与展望

尽管系统已实现预期功能，但仍存在以下优化空间：

**功能扩展**

实验资源管理：增加实验文件上传 / 下载、在线编辑功能，支持代码文件自动评测（如 Python 代

**统计分析**：引入数据可视化模块（如 Matplotlib/Pandas），生成实验完成率、成绩分布等图表，辅助

数据库扩展：支持 MySQL

**操作反馈**：增加加载动画、批量操作（如批量批改作业、批量导入学生信息）和快捷键支持，减少用户重复操作。

**数据备份与恢复**：实现自动定时备份数据库，并提供可视化的备份管理界面。

参考文献

1. <https://blog.csdn.net/qq_49443542/article/details/145267832>
2. [PyQt5/PyQt: PyQt Examples（PyQt各种测试和例子） PyQt4 PyQt5](https://github.com/PyQt5/PyQt)
3. [Python logging 模块 | 菜鸟教程](https://www.runoob.com/python3/python-logging.html)
4. [SQLite 教程 | 菜鸟教程](https://www.runoob.com/sqlite/sqlite-tutorial.html)
5. [【QT】史上最全最详细的QSS样式表用法及用例说明\_qt qss-CSDN博客](https://blog.csdn.net/WL0616/article/details/129118087)