北京电子科技学院（BESTI）

**实 验 报 告**

课程：信息安全系统设计 班级：2214

姓名：徐鹿鸣 学号：20221414

成绩： 指导教师：娄嘉鹏 实验日期：2024/12/08

实验密级：无 预习程度：全预习

实验时间：3，4节 仪器组次：无 必修/选修：必修

实验序号：实验四

实验名称： 密码模块的应用

实验目的与要求：基于商用密码标准的密码模块的应用。对电子公文系统等密码系统进行威胁建模，安全性分析和安全性设计。

实验仪器：

|  |  |  |
| --- | --- | --- |
| 名称 | 型号 | 数量 |
| 计算机 |  | 1 |
|  |  |  |

实验内容、步骤与体会（附纸）：

**1. 实验内容**

实现电子公文系统。要求：

1.系统采用B/S，或者C/S架构，参考《GM∕T 0054-2018 信息系统密码应用基本要求》和《GB/T 39786-2021《信息安全技术 信息系统密码应用基本要求》》

2.设计实现的系统管理功能要简单易用；

3.用户管理采用“三员”管理；

4.设计实现的密钥管理功能应覆盖密钥的全生命周期；

**2. 实验步骤**

下面的步骤与实际的过程会有一定的出入，因为中途肯定是有很多徘徊和退后的。我将简述我是如何学习并实现这个项目的。

在实践过程中极其大量地使用了AI来帮助搜集资料、学习功能、代码编写等，同时也参考了一些课程与开源demo来辅助学习。

**2.1步骤1 前期准备与功能测试**

**2.1.1 操作内容**

考虑到学习难度，决定使用python-flask框架实现电子公文系统。同时对一些可能用到的功能进行验证，包括国密算法的使用、邮箱发送验证码（未应用）、强制HTTPS访问、数据库连接等。

功能验证其实是一直在做的，一方面是先在“外面”测试成功才可能可以集成到实际项目中来，另一方面也是验证技术路线是否可行，比如通过测试发现在web端访问KEY基本上不是我个人能够实现的（尽管官方说支持浏览器），所以放弃了由KEY来验证用户身份的想法。

验证的脚本集中放置在项目的“一般功能验证”文件夹下。

**2.1.2操作结果**

****

图 1 截至实验报告撰写时的功能验证

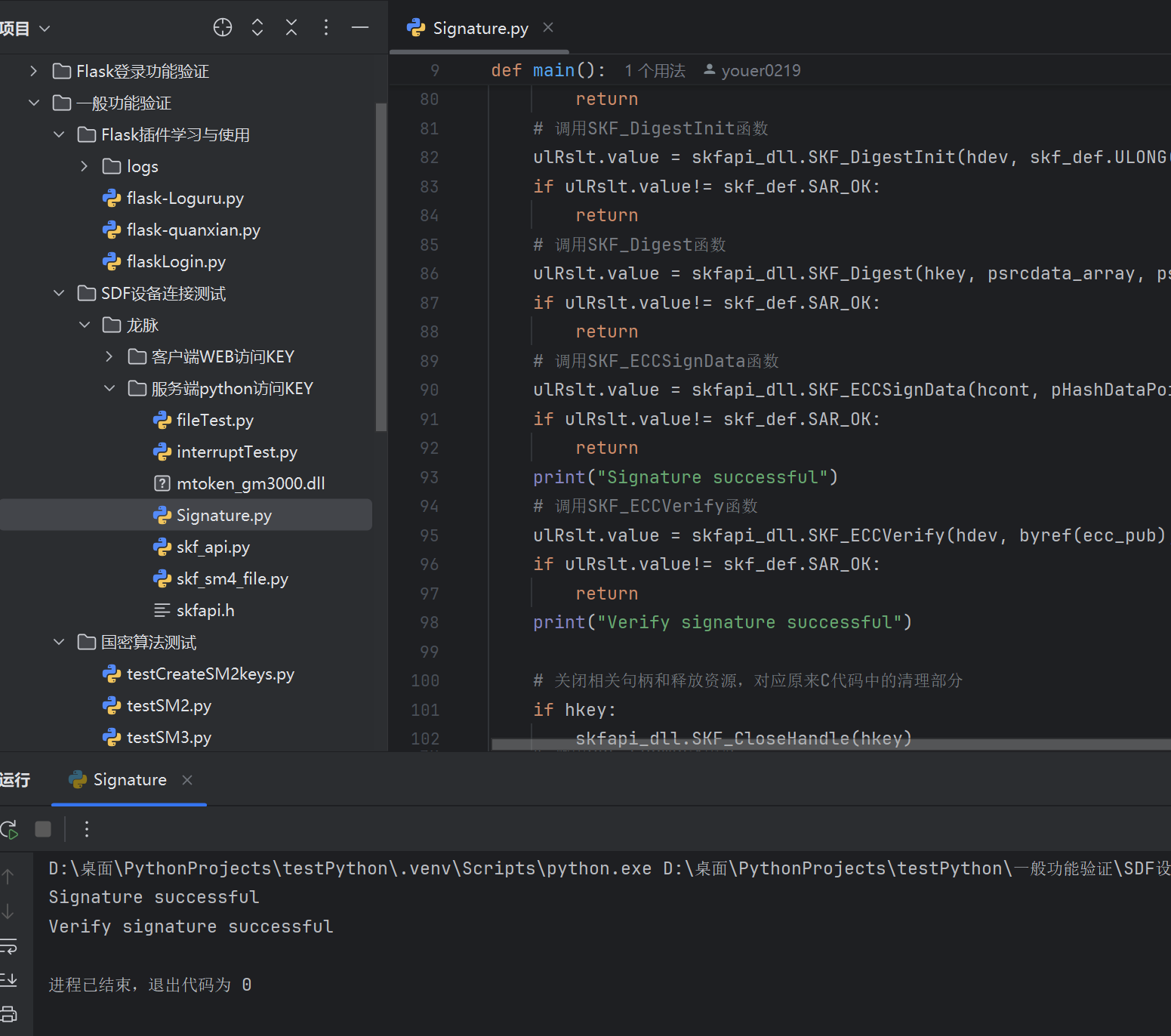
****

图 2 龙脉KEY的签名demo的python实现与测试

**2.2步骤2 实现登录系统**

**2.2.1 操作内容**

学习B站课程[python flask连接mysql数据库实现登录注册](https://www.bilibili.com/video/BV1P5x7emE9H/?spm_id_from=333.337.search-card.all.click&vd_source=912de37828db7e4feff5c9492864d51c)实现最基础的登录操作。然后修改用户表数据结构，应用SM3哈希算法，实现带盐值的登录验证。之后再学习flask-login插件的使用并将其集成到项目中。

在这里，我学到了最基础的登录登出、用户注册流程实现与数据库的操作（查询、插入、修改）。同时flask-login插件的使用实现了登录保护功能，并很好的增强了项目的可扩展性，为之后的实践打好了基础。

项目放在了“Flask登录功能验证”文件夹中，后续实际开发时就是以这个项目为基础进行开发的。

**2.2.2操作结果**

****

图 3 最初的登录界面

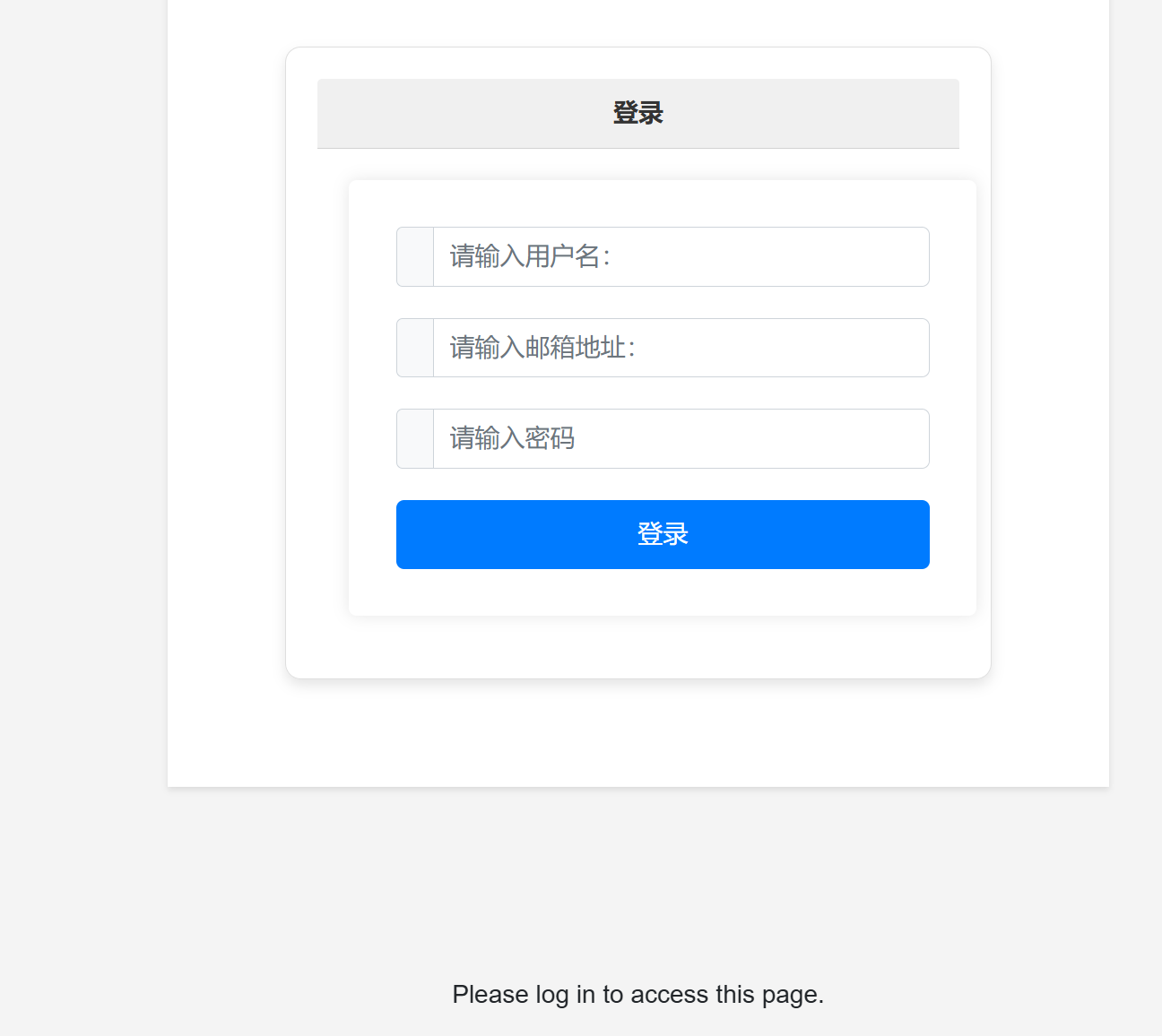


图 4 目前的登录保护（下面的提示）

**2.3步骤3 实现博客系统**

**2.3.1 操作内容**

博客系统是为公文系统做准备的。我们实验的要求是公文系统建模，我当时认为博客系统对自己博客的发布、编辑、删除等操作与公文相关操作是类似的，所以学习了B站课程：[90分钟用Python和Flask开发博客系统](https://www.bilibili.com/video/BV1gh411q7xo/?spm_id_from=333.337.search-card.all.click&vd_source=912de37828db7e4feff5c9492864d51c)，同时发现了一个开源的[基于flask的对用户增删改查的项目](https://github.com/mudfish/python-flask-user-crud)。

由此有了数据库操作的基本代码模板和前端显示的基本代码模板，包括导航页的设置、博客列表的实现，以及学会如何设置各种路由并实现路由的跳转等，至少会搭建一个基本框架然后让AI来填充内容。

博客项目放在了“Flask博客系统”文件夹下。

**2.3.2操作结果**

****

图 5 博客页面

**2.4步骤4 开始开发与公文操作实现**

**2.4.1 操作内容**

在上周日准备相对充分时开始了实际项目的构建。选择以登录项目为后端代码模板，以博客系统的html文件为前端代码模板正式开始开发。

先从对公文的数据结构建模开始，再显示到前端的公文列表中，再实现筛选公文列表的公文一步步来。这里的结构是很相似的，一般先实现一个列表展示可操作的对象，然后进行查看，在只可读公文的查看界面中进行编辑或审核或签收操作，编辑操作在单独的编辑界面中进行。

这里以demo01数据库模拟本地公文数据库，以public数据库模拟外部环境。暂时以明文存储各种数据。

**2.4.2操作结果**

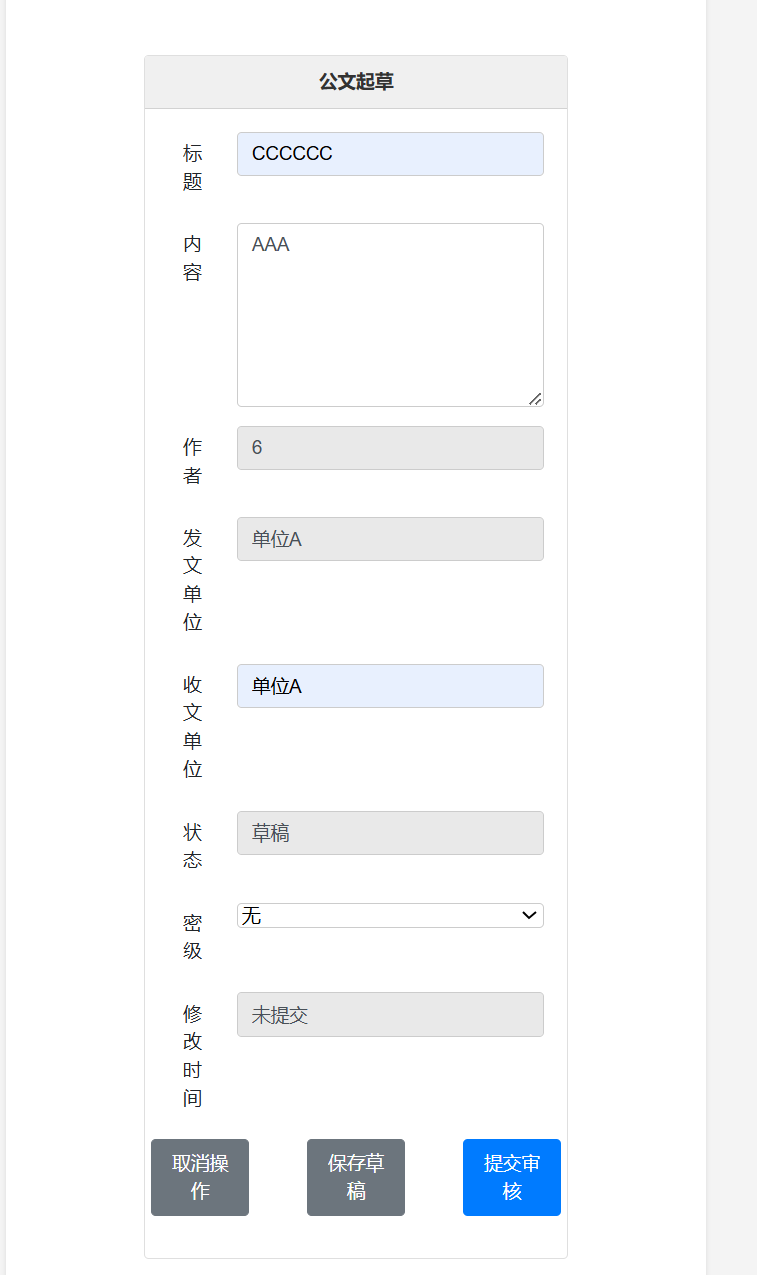
****

图 6 公文起草功能

****

图 7 公文签收功能

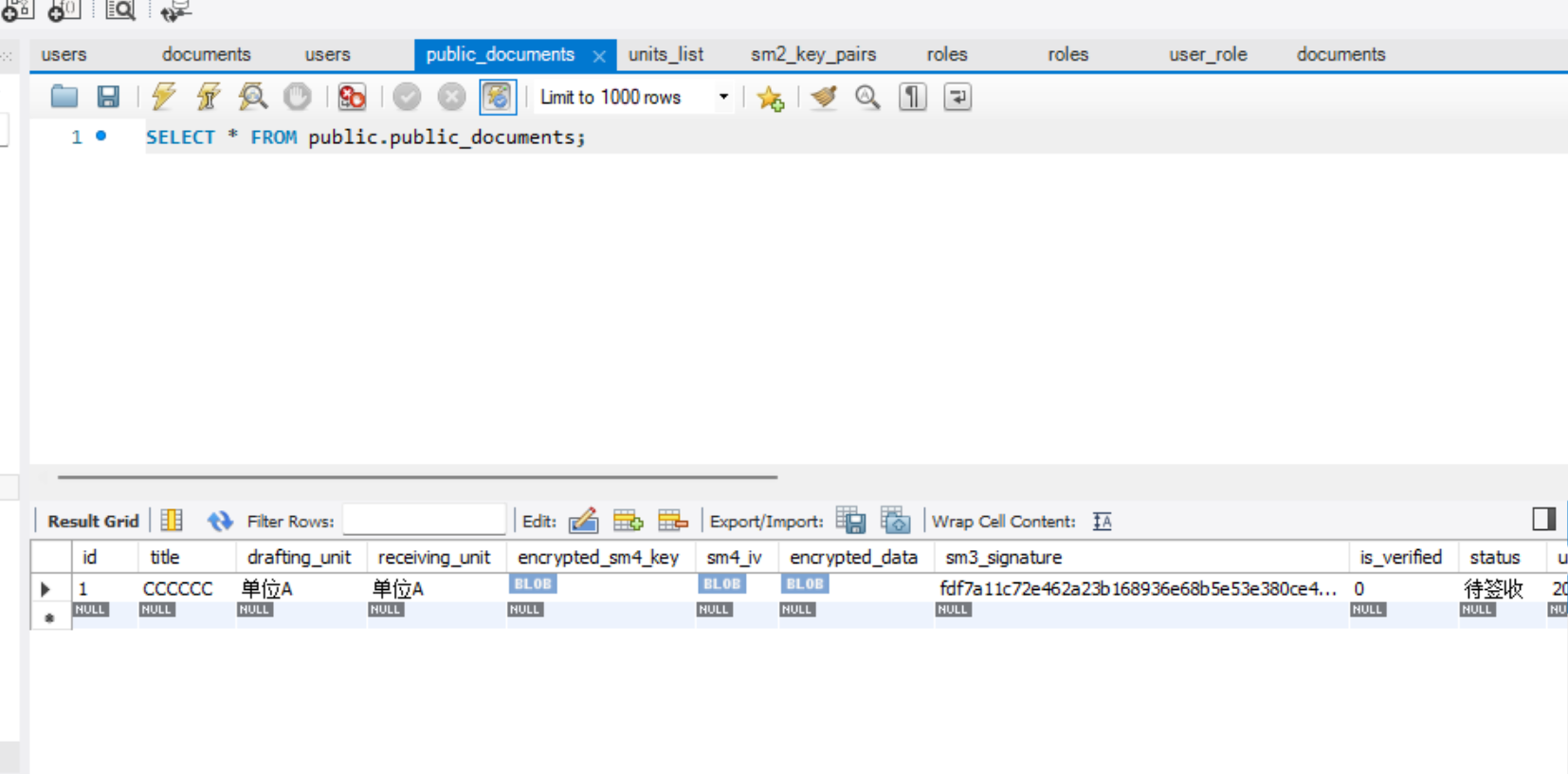
****

图 8 发送的公文的数据库存储状态（这是后期有加密功能后的数据库存储）

**2.5步骤5 通过装饰器实现本地公文和收发公文的密码应用**

**2.5.1 操作内容**

实现两个装饰器，应用在在数据库中执行语句获取结果的con-my-sql函数上。

在本地公文装饰器中，对于输入，装饰器会将公文内容字段提取出来进行加密，返回加密后的数据和SM4加密使用的随机生成的IV值，将其保存到公文输入字典中。对于输出，会判断是否为列表，遍历列表中的每一个字典，将加密的内容和SM4-iv值提出出来用来解密，将解密后的内容保存到输出的字典中。

在发送与接受的公文的装饰器中，对于输入，将要发送的公文字典转换成JSON格式在转换成为比特进行处理，返回加密后的数据、被对方公钥加密的临时生成的SM4密钥值、SM4iv值、对公文数据哈希值的SM2签名私钥签名值（hex格式）。将其一一保存到输入的字典中。对于输出，遍历列表中的每一个字典，用自己的解密私钥进行解密，用解密后的SM4密钥和SM4-IV值解密公文，用对方的SM2验签公钥验证签名。将验签结果、解密后的结果等都一一保存到输出字典中。

此时的SM2公私钥对和SM4密钥暂时明文硬编码。

**2.5.2操作结果**

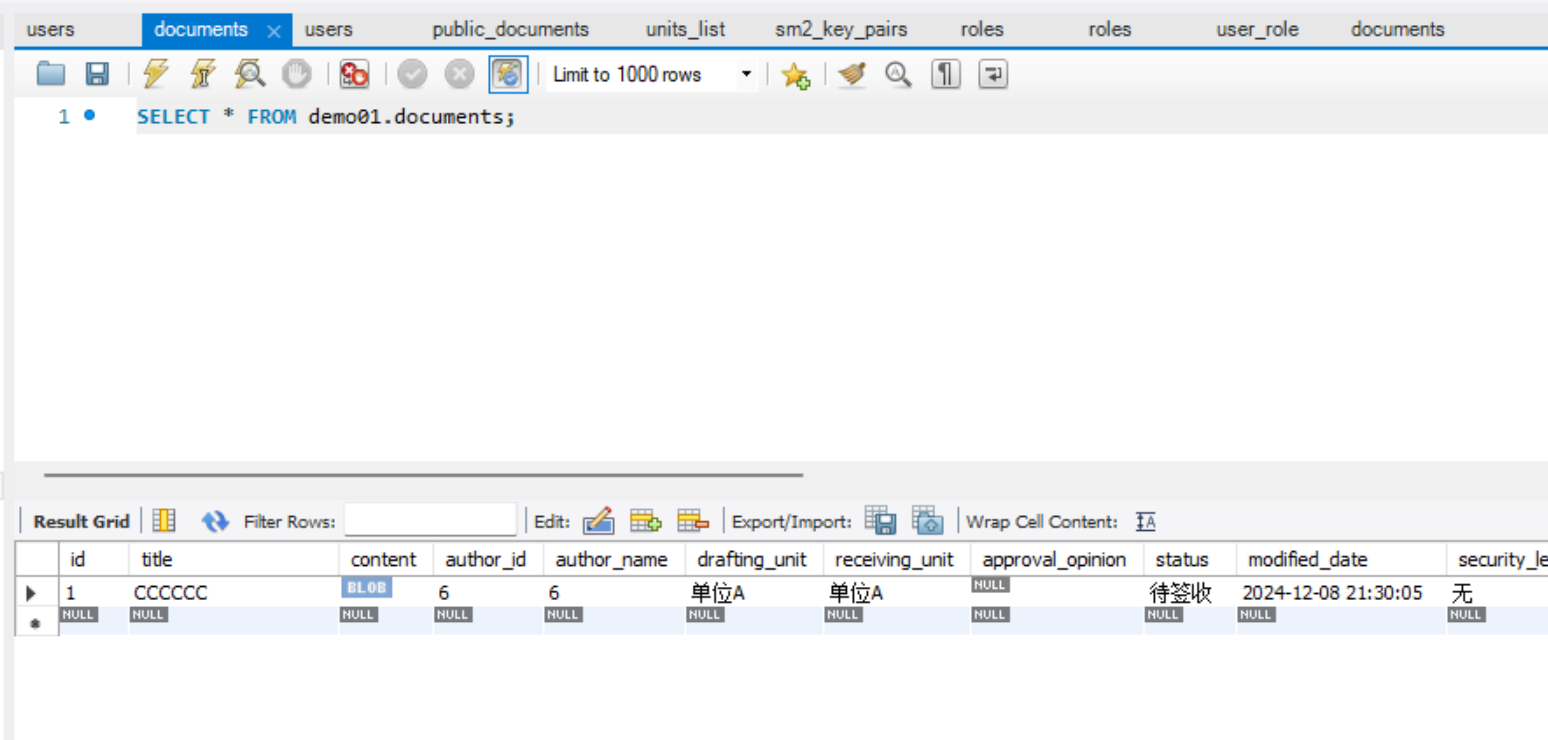
****

图 9 本地公文加密存储

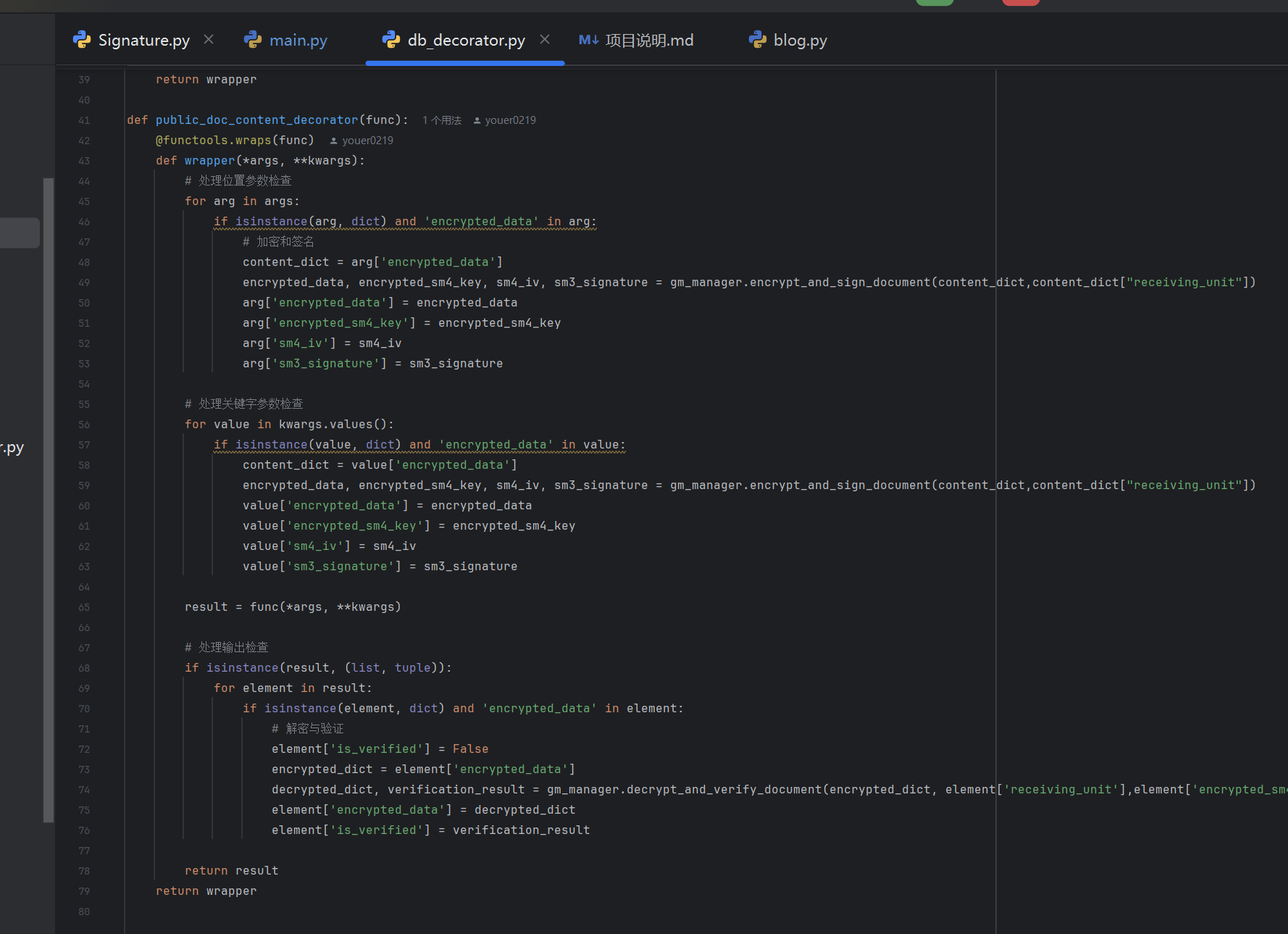
****

图 10 收发公文的装饰器函数（两个都在db\_decorator.py中）

**2.6步骤6 实现基于硬件的SM4密钥保存与应用**

**2.6.1 操作内容**

在之前的功能验证中，已经实现了签名demo的python代码。通过阅读demo代码和头文件，我理解了对于KEY而言，需要先找到设备、打开设备、打开应用、打开容器才能访问到具体的内容。前三个步骤直接使用demo中的代码就行了。对于容器，其头文件中说只支持ECC和RSA容器，所以要想保存SM4密钥需要使用文件。

那么，为什么不直接保存私钥呢？理论上讲肯定要保存私钥，然后在KEY中模拟在硬件中进行私钥签名和解密，安全性显然更高。但在头文件中没有直接的私钥解密的API，可能需要通过密钥协商等措施进行。这方面过于复杂，时间太少，所以放弃。

通过fileTest.py的测试，我实现了对文件的创建、删除、写入和读取操作。尽管有些东西比如权限的设置没搞明白（直接写个255），但不影响我实际使用。

在这种密码架构中，我需要讲自己的公私钥对保存在数据库中，其中私钥都要使用SM4密钥进行加密存储。这个也是通过装饰器实现的，与本地公文的装饰器原理一致。

添加硬件后项目会明显卡顿，所以一般测试时会不使用硬件而是直接以硬编码的形式返回需要的SM4密钥值。实际应用中肯定也有更专业的设备于解决方案。

从中可以看出，数据库部分依赖密码部分，密码部分也依赖数据库部分，这就造成了循环引用问题。这个问题的解决请参考第三部分的内容。

**2.6.2操作结果**

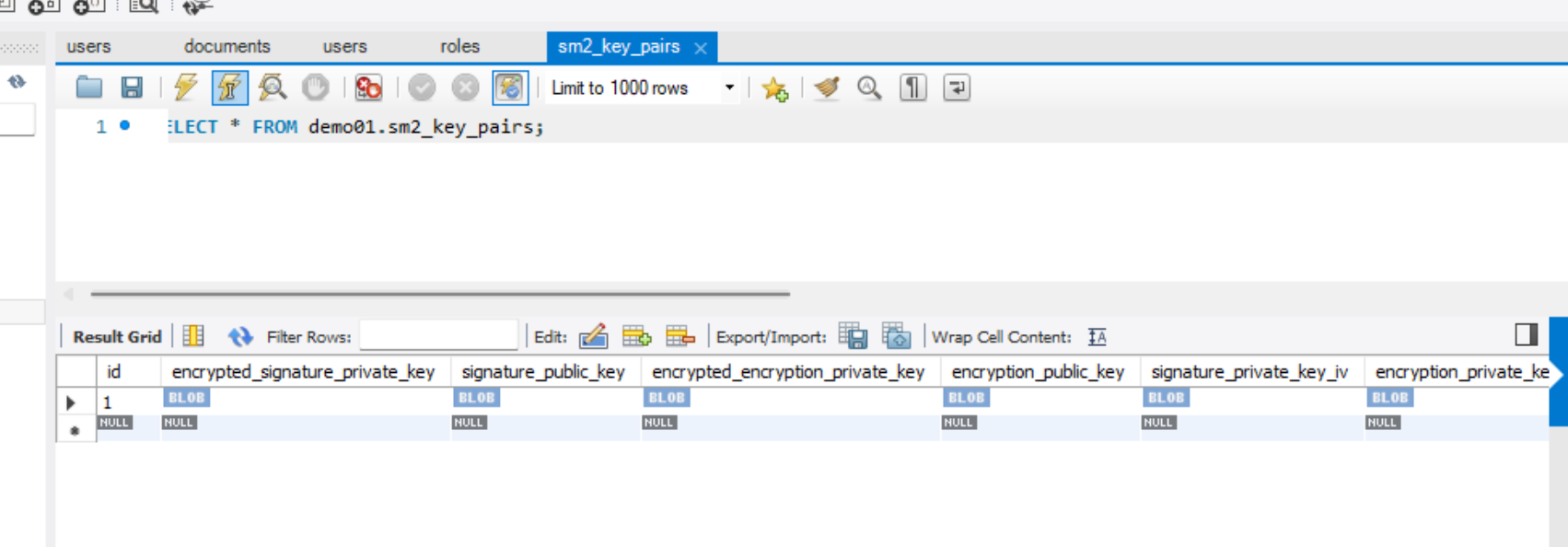
****

图 11 在数据库中存储私钥密文和公钥明文

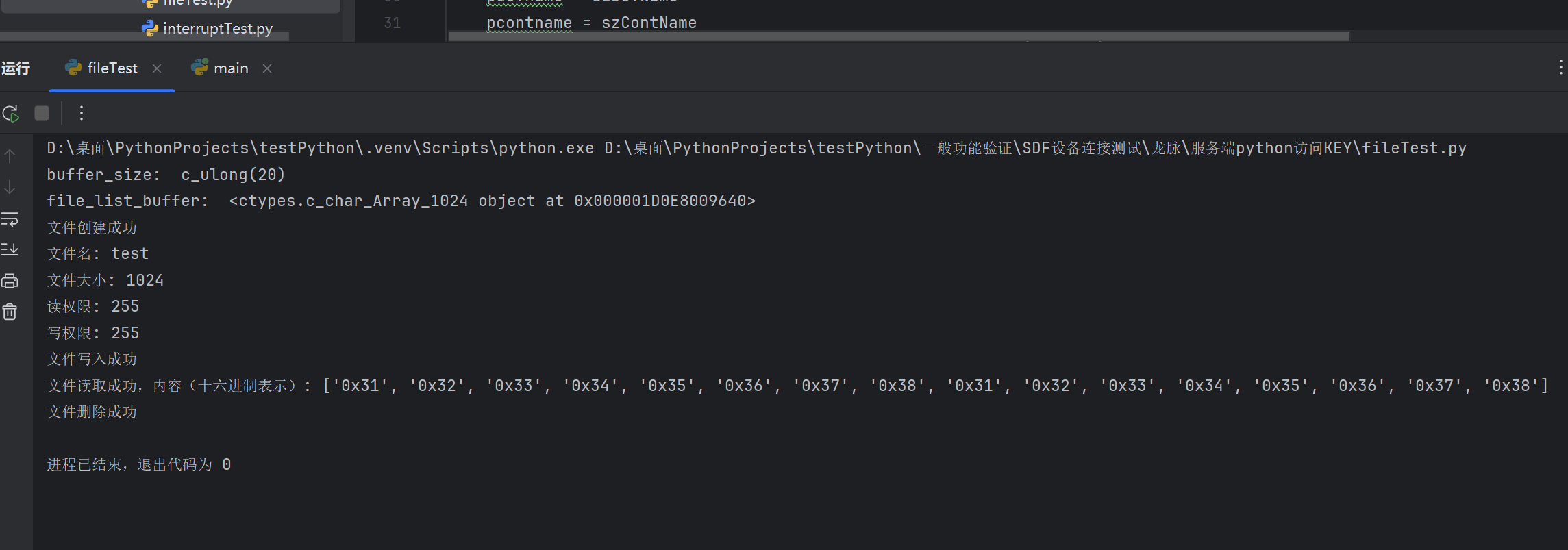
****

图 12 fileTest.py测试结果

**2.7步骤7 实现用户管理、日志管理和权限管理**

**2.7.1 操作内容**

用户管理与之前的公文的增删改查并无不同。这里不表。

日志管理使用了loguru库，只简单现了对用户操作的记录和在前端展示日志内容并以颜色区分不同级别。分页功能、查找筛选功能以及统计功能都没有实现。

权限管理使用了flask\_principal插件，同时新设计了角色表、用户-角色表、权限表、角色-权限表来实现基于角色的访问控制。所有因权限访问失败的用户行为将被记录日志。但只实现了基本的框架，没有实现权限的添加删除与其他的必要限制。

**2.7.2操作结果**

****

图 13 用户管理（新增用户）

****

图 14 日志查看

****

图 15 权限管理（只实现了列表）

这里的用户名6的用户是用来测试的“超级用户”

**2.8步骤8** 系统量化评估

**2.8.1 操作内容**

按照[商用密码应用安全性评估量化评估规则](https://ht.cacrnet.org.cn/upload/file/20230718/1689639379401088.pdf),计算自己系统的得分，只计算应用和数据安全。

评分结果：

1. 身份鉴别：0.5 只实现了带盐值的哈希密码盐值。但密码强度低。认证失败无惩罚。
2. 访问控制信息完整性：0.5 权限划分颗粒度还是不够细。有日志但日志可篡改。没有实现权限变更功能。
3. 重要信息资源安全标记完整性：0.5 有对公文密级、单位的标注但没有留痕机制
4. 重要数据传输机密性：1 强制HTTPS访问网站并加密数据、公文传输采取数字信封模式使用临时SM4密钥加密保护
5. 重要数据存储机密性：1 使用硬件保存加密密钥、使用SM4算法加密存储本地公文数据、不明文存储用户口令
6. 重要数据传输完整性：0.5 公文传输有对SM3哈希的签名值的生成与验证
7. 重要数据存储完整性：0 没有对本地存储的公文数据的哈希值验证 没有数据库备份功能
8. 不可否认性： 1 单位收发公文使用自己的私钥，不可否认

综合得分：5分

**2.8.2操作结果**

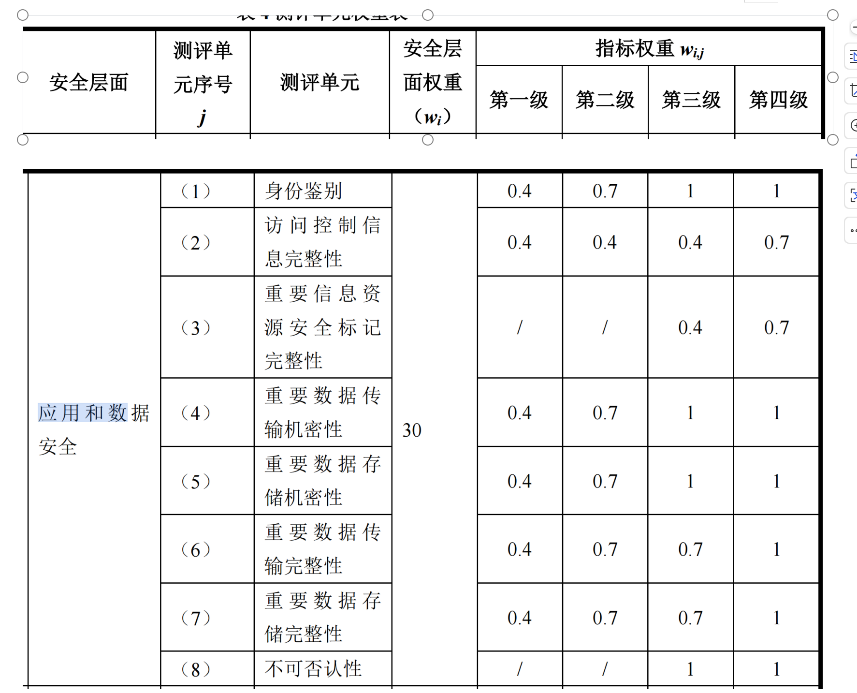
****

图 16 量化评分标准

**3．实验体会**

**3.1** 调试中出现的问题及解决过程

**3.1.1** 循环引用问题

当时主要是数据库功能和密码功能两者相互依赖。由于文件太多看不过来，所以以main为起点画了依赖关系图帮助理解。直线是直接依赖，虚线是间接依赖。

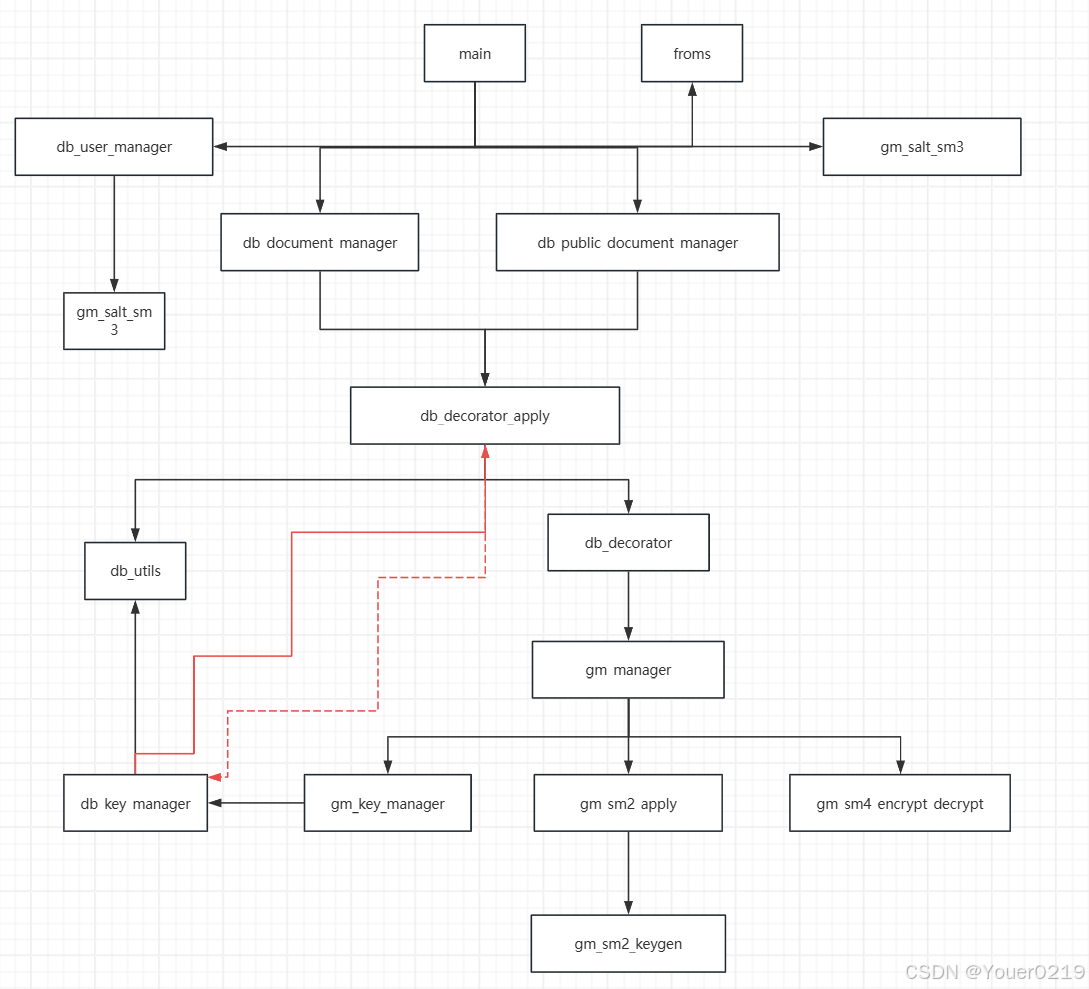


图 17 依赖关系图

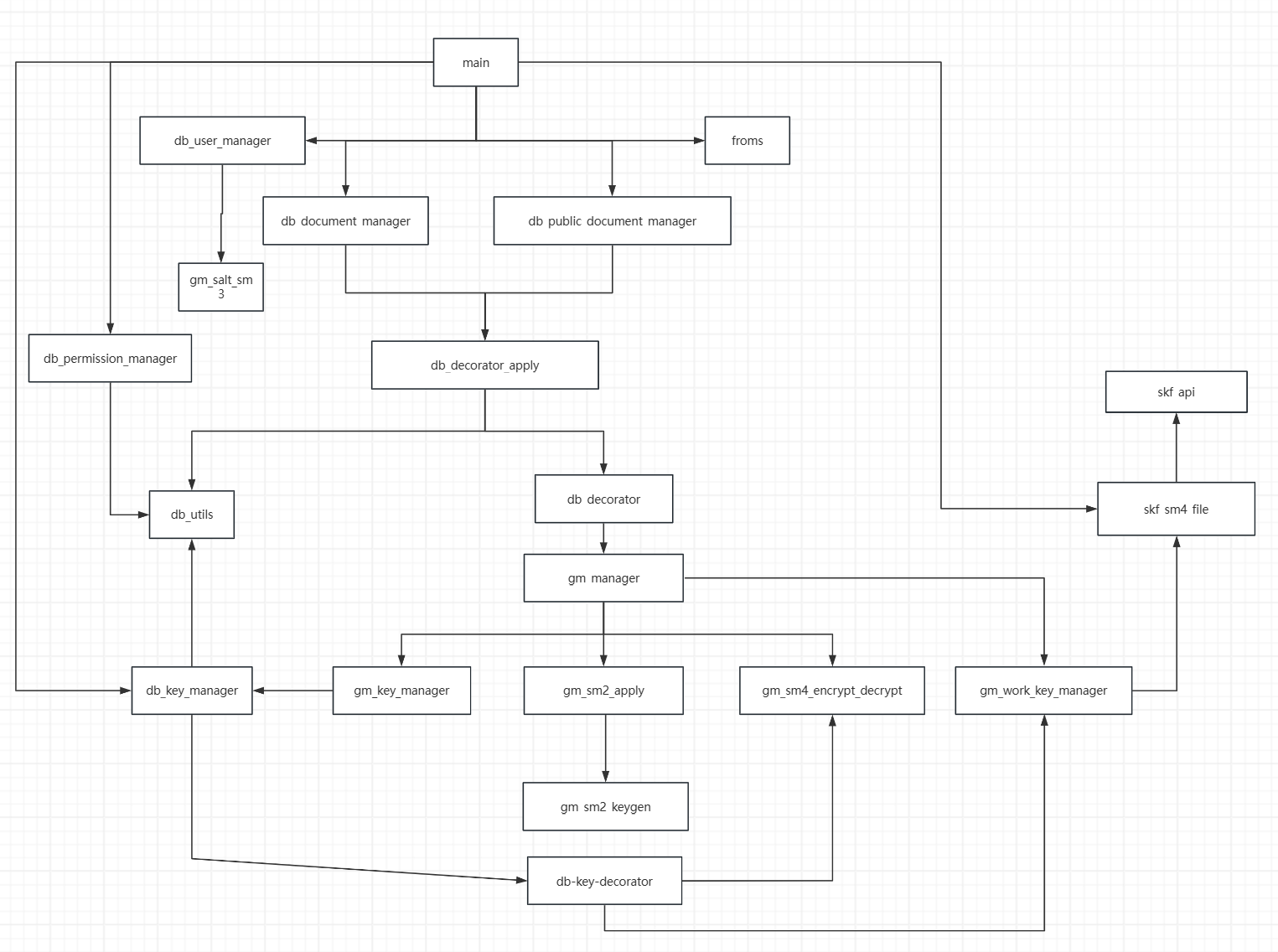
最后是将db-key-manager（用来获取SM2公私钥对）与db-decocator-apply（提供装饰器代码）的进行解耦合。最终效果如下：  


图 18 最终的依赖关系图

**3.1.2** 集成权限管理插件中数据库莫名理解失败与CSS文件失效问题

这个问题的原理我没有搞明白，但触发条件大概是在权限插件的# 身份加载事件处理函数中使用了登录插件中的current\_user变量，就会导致数据库连接极端不稳定同时多次连接后会失败。同时前端界面的样式基本上都失效了。

由于权限设置是通过数据库获取的用户角色信息，所以同时会导致用户权限失效，导致合法访问被拒绝。

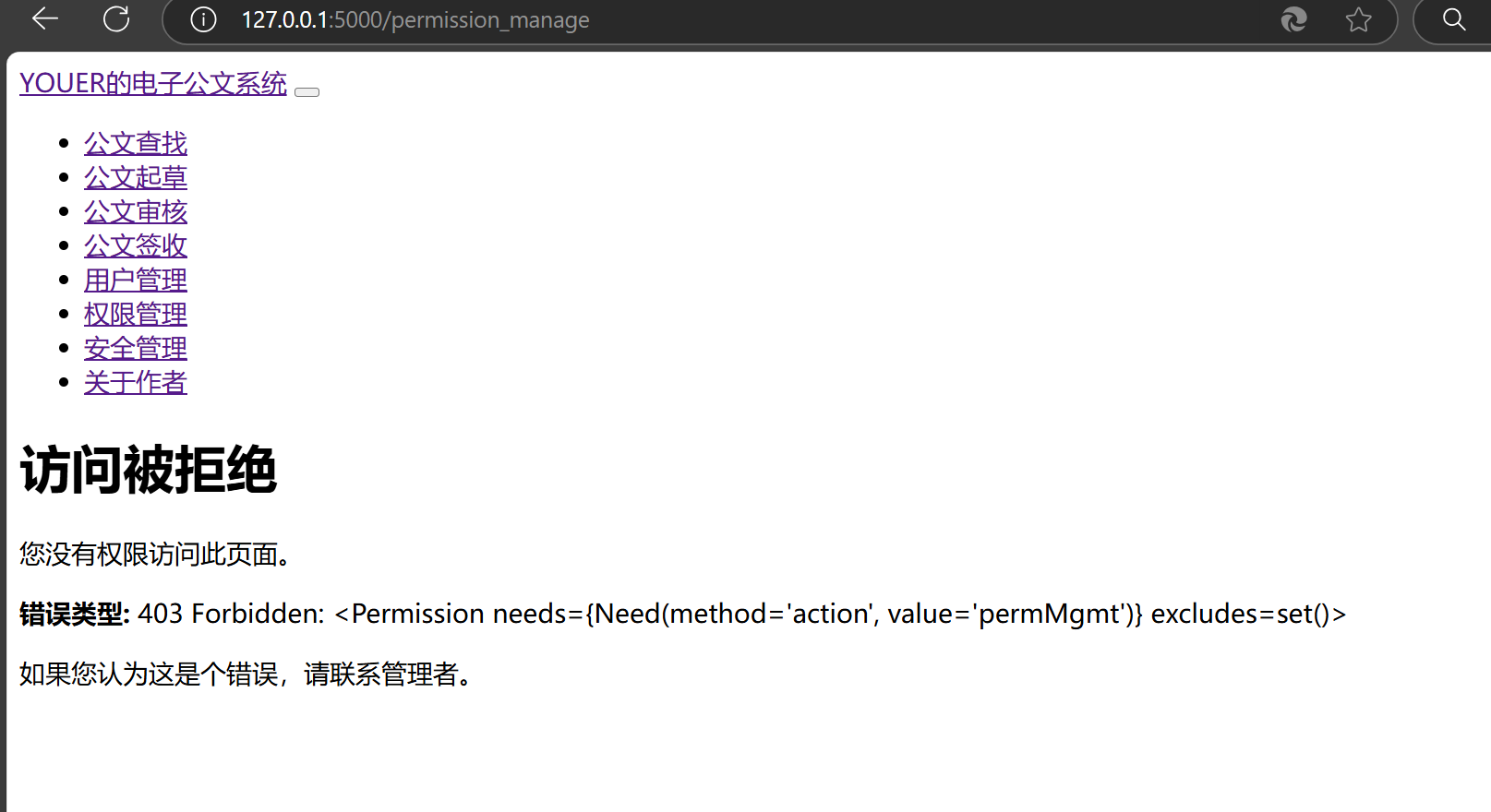


图 19 报错样式

那么，在该函数中直接与数据库连接是否可行呢？测试结果是，很不稳定。控制台会有很多报错信息，包括发的包的顺序不对等，有时也会导致权限设置失效。

考虑到他在login函数中访问数据库一般没有问题，所以我的策略就是在登录时获取用户权限信息，将其保存到字典current\_user\_permission中，用户可以以自己ID来获取权限信息并进行测试。当用户登出时就清空该字典。

如果后续有用户权限发生变动，可以再设置一个让用户强制登出的环节更新权限信息。

**3.2 心得体会**

过于劳累。

**3.3 实验改进建议**

两周内实现这个系统还是太有挑战性了。找学长也不是很方便。建议保留一些学长学姐的优秀作业来学习参考并修改。