

# 计算机网络安全实验报告

实验名称：完整性访问控制系统设计与实现

班级：1703101

学号：1170300520

姓名：郭子阳

计算机学院

# 1. 实验环境

- macOS 10.15.2
- Node.js 12.13.1
- Mysql Community 8.0.18
- Electron

# 2. 实验成果

## 2.1 给出应用系统的安全策略文档

在登陆界面，可以进行登陆操作，系统通过对数据库的查询来确认登陆者是普通用户还是管理员，并跳转到对应的操作界面。

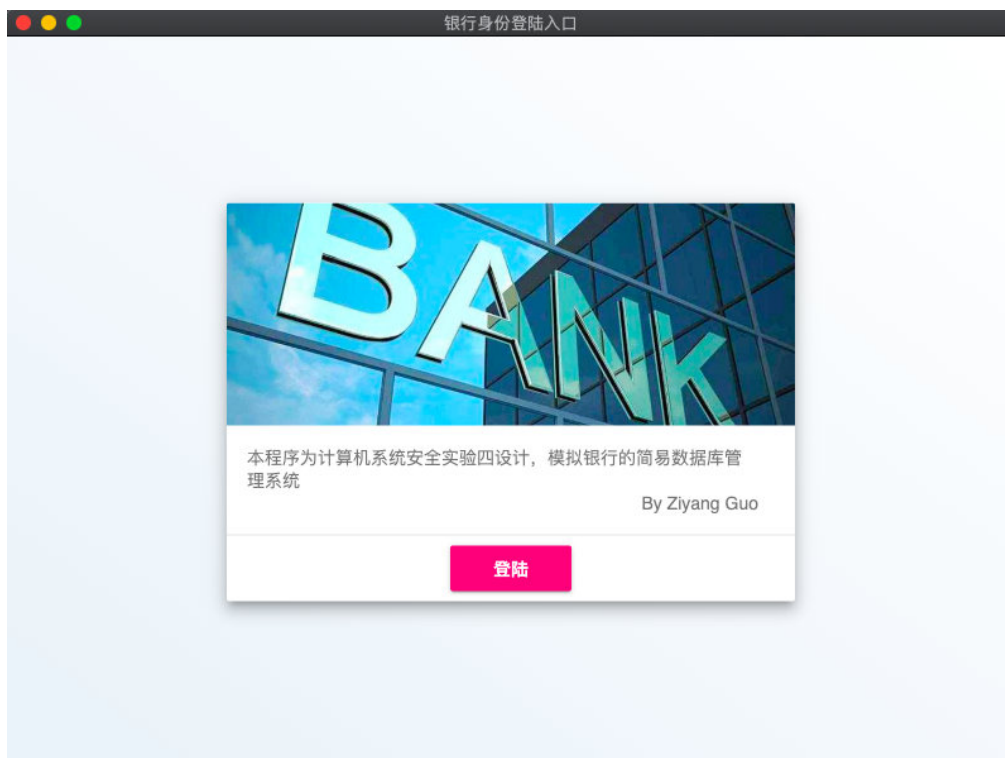
普通用户可以进行申请存款、取款和查询余额的操作。

当用户申请存款或取款时，首先需要输入存/取款的金额，接着会生成一个请求发往在线的管理员端，请求中包含用户的用户名信息和存/取款金额，管理员端收到请求后，会首先向数据库查询用户余额，如果是取款操作但是余额不足时，会直接拒绝用户请求，否则，会在界面上显示请求内容，若管理员同意，管理员会更新数据库，并将操作结果返回给用户，否则会向用户返回拒绝。

查询余额时，用户无需向管理员请求，普通用户拥有余额表的查询权限。

## 2.2 提供交互界面，能够完成录入、查询等功能

首页：



登陆界面：



用户操作界面：



管理员操作界面：



用户发起请求界面：



管理员接受请求界面：



管理员拒绝用户操作后的提示：



## 2.3 满足责任分离、功能分离原则

责任分离原则要求：需要多步完成一个事务时，需两个以上人员共同完成。存款和取款会对数据库操作，需要满足责任分离原则，需要用户和管理员共同操作，用户发起请求，管理员同意请求，只有用户或只有管理员都无法完成存款和取款操作，没有办法修改余额的数值，满足责任分离原则，保证了数据的完整性。

功能分离原则要求：开发数据不应该污染产品数据。开发时采用了虚拟机环境，并且导出了数据库以备份，防止对产品数据的污染。

## 2.4 保存审计日志

系统日志保存在log文件夹下，以系统启动时间的时间戳为文件名。日志分为INFO、WARN和ERROR级别，记录了从系统启动到系统关闭的所有操作。

以下日志以用户ziyang申请存钱100元、管理员同意、申请取钱30元、管理员拒绝为例：

用户ziyang端日志：

1	[2019-12-21T19:39:33.972] [INFO] bank - 启动登陆页
2	[2019-12-21T19:40:38.034] [INFO] bank - 用户 ziyang 登陆成功!
3	[2019-12-21T19:40:46.769] [INFO] bank - 用户 ziyang 尝试存入 100 元
4	[2019-12-21T19:40:48.796] [INFO] bank - 用户操作成功!
5	[2019-12-21T19:40:51.656] [INFO] bank - 用户 ziyang 尝试取出 30 元
6	[2019-12-21T19:40:53.176] [WARN] bank - 用户操作失败! 操作被管理员拒绝
7	[2019-12-21T19:40:55.561] [INFO] bank - 管理员 ziyang 退出登陆
8	[2019-12-21T19:41:01.324] [INFO] bank - 退出程序

管理员admin端日志：

1	[2019-12-21T19:39:38.109] [INFO] bank - 启动登陆页
2	[2019-12-21T19:40:32.980] [INFO] bank - 管理员 admin 登陆成功!
3	[2019-12-21T19:40:33.759] [INFO] bank - 服务器监听端口 8080
4	[2019-12-21T19:40:46.775] [INFO] bank - 获得用户 ziyang 请求 存入 100 元
5	[2019-12-21T19:40:48.794] [INFO] bank - 用户 ziyang 存入 100 成功! 余额 530 元
6	[2019-12-21T19:40:51.658] [INFO] bank - 获得用户 ziyang 请求 取出 30 元
7	[2019-12-21T19:40:53.175] [INFO] bank - 管理员拒绝了用户 ziyang 的操作
8	[2019-12-21T19:40:57.231] [INFO] bank - 用户 admin 退出登陆
9	[2019-12-21T19:41:00.167] [INFO] bank - 退出程序

## 2.5 遵循Clark-Wilson模型，定义应用系统的完整性限制条件

Clark-Wilson模型考虑如下几点：

1. 主体必须被识别和认证
2. 客体只能通过规定的程序进行操作
3. 主体只能执行规定的程序
4. 必须维护正确的审计日志
5. 系统必须被证明能够正确工作

程序以管理员为主体，普通用户为客体，管理员登陆的过程即为识别和认证的过程，通过识别和认证的管理人员才可以执行管理员权限下的操作，满足第一条。

普通用户作为客体只可以执行申请存款取款、查询余额的操作，无法进行其他操作，满足第二条。

主体只可以同意或拒绝用户操作，无法独自进行这些操作，满足第三条。

通过以上的操作，可以确定程序维护了正确的审计日志可供审计检查。且程序运行正确。满足第四条和第五条。

## 2.6 遵循Clark-Wilson模型的证明规则和实施规则

**CR1:** 当任意 IVP 运行时，它必须保证所有的CDI处于有效状态

**体现:** 用户登陆后，在没有管理员同意的情况下，无法操作数据库中的余额数据，只能向管理员申请，管理员同意后才可以修改。

**CR2:** 对相关联的CDI，一个TP必须将这些CDI从一个有效状态转到另一个有效状态

**体现:** 管理员同意或拒绝用户的操作后，用户的请求就转换为两种状态：成功状态和失败状态。

**ER1:** 系统要维护关联关系，保证经过验证的TP操作相应的CDI

**体现:** 用户提出的存取款请求被管理员同意后，则该请求就经过了验证，经过验证的请求才可以发挥相应的作用，完成对应的操作，修改对应账户的余额

**ER2:** TP操作CDI时，保证操作用户有权对相应CDI做操作，TP所代表的用户是CDI的真实用户

**体现:** 管理员同意的请求才可以对数据库操作

**CR3:** 系统执行操作时，符合责任分离原则

**体现:** 系统分为两种角色，普通用户和管理员，修改数据库的操作需要两个角色同时操作，任何单方面操作都无法执行操作

**ER3:** 系统执行TP前，应验证用户身份

**体现:** 用户发起请求前，需要首先登陆，验证用户的身份

**ER4:** 只有可以授予TP访问规则的主体才能修改列表中相应的表项，授权主体不能执行TP操作

**体现:** 用户提出修改余额的申请，只有用户才可以修改对应账户的余额，授权的主体是管理员，无法执行请求对应的操作

### 3. 系统设计

系统使用了三个表：bankuser、bankcurrency和bankidentity表，表结构如下：

bankuser:

列名	数据类型	含义
id	int	用户唯一ID
username	varchar(255)	用户名
valid	tinyint(1)	用户账户是否有效

bankcurrency:

列名	数据类型	含义
id	int	用户ID
currency	int	用户账户余额

bankidentity:

列名	数据类型	含义
id	int	用户ID
isadmin	tinyint(1)	用户是否是管理员

同时，创建了四个存储过程来简化操作，且可以通过存储过程的权限设置约束用户行为。存储过程如下：

过程名称	过程参数	过程含义
queryalluser	无参	管理员获取所有用户的信息
changeurrencybyusername	用户名、新的账户余额	修改用户的余额为参数余额
querysingleusercurrency	用户名	通过用户名获取用户余额
querysingleuseridentity	用户名	通过用户名获取用户身份

整个数据库建库语句如下：

```
1 DROP DATABASE IF EXISTS lab4;
2 CREATE DATABASE `lab4` DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
3 USE lab4;
4
5 CREATE TABLE `bankuser`
6 (
7     id int PRIMARY KEY,
8     username varchar(255) UNIQUE,
9     valid tinyint(1) NOT NULL
10 );
11
12 CREATE TABLE `bankcurrency`
13 (
14     id int PRIMARY KEY,
15     currency int NOT NULL,
16     FOREIGN KEY(id) REFERENCES bankuser(id)
17 );
18
19 CREATE TABLE `bankidentity`
20 (
21     id int PRIMARY KEY,
22     isadmin tinyint(1) NOT NULL,
23     FOREIGN KEY(id) REFERENCES bankuser(id)
24 );
25
26 INSERT INTO `bankuser` VALUES (1, 'admin', true);
27 INSERT INTO `bankuser` VALUES (2, 'ziyang', true);
28 INSERT INTO `bankuser` VALUES (3, 'exp', true);
```



```

29
30 INSERT INTO `bankcurrency` VALUES (2, 100);
31 INSERT INTO `bankcurrency` VALUES (3, 40);
32
33 INSERT INTO `bankidentity` VALUES (1, 1);
34 INSERT INTO `bankidentity` VALUES (2, 0);
35 INSERT INTO `bankidentity` VALUES (3, 0);
36
37 DROP USER IF EXISTS 'admin'@'localhost';
38 DROP USER IF EXISTS 'ziyang'@'localhost';
39 DROP USER IF EXISTS 'exp'@'localhost';
40
41 CREATE USER 'admin'@'localhost' IDENTIFIED WITH mysql_native_password BY
    'admin';
42 CREATE USER 'ziyang'@'localhost' IDENTIFIED WITH mysql_native_password BY
    'ziyang';
43 CREATE USER 'exp'@'localhost' IDENTIFIED WITH mysql_native_password BY 'exp';
44
45 DROP PROCEDURE IF EXISTS queryalluser;
46 DELIMITER $$
47 CREATE PROCEDURE queryalluser()
48 BEGIN
49     SELECT lab4.bankuser.id, lab4.bankuser.username, lab4.bankcurrency.currency
50     FROM lab4.bankuser, lab4.bankidentity, lab4.bankcurrency
51     WHERE lab4.bankuser.id = lab4.bankcurrency.id
52     AND lab4.bankuser.id = lab4.bankidentity.id
53     AND lab4.bankidentity.isadmin = 0
54     AND lab4.bankuser.valid = true;
55 END$$
56 DELIMITER ;
57
58 DROP PROCEDURE IF EXISTS changecurrencybyusername;
59 DELIMITER $$
60 CREATE PROCEDURE changecurrencybyusername(IN p_username varchar(255), IN
    p_currency int)
61 BEGIN
62     DECLARE p_id int;
63     SELECT id INTO p_id FROM lab4.bankuser WHERE username=p_username AND
        valid=true;
64     UPDATE lab4.bankcurrency SET currency=p_currency WHERE id=p_id;
65 END$$
66 DELIMITER ;
67
68 DROP PROCEDURE IF EXISTS querysingleusercurrency;
69 DELIMITER $$
70 CREATE PROCEDURE querysingleusercurrency(IN p_username varchar(255))
71 BEGIN
72     DECLARE p_id int;

```

```

73     SELECT id INTO p_id FROM lab4.bankuser WHERE username=p_username AND
valid=true;
74     SELECT currency FROM lab4.bankcurrency WHERE id=p_id;
75 END$$
76 DELIMITER ;
77
78 DROP PROCEDURE IF EXISTS querysingleuseridentity;
79 DELIMITER $$
80 CREATE PROCEDURE querysingleuseridentity(IN p_username varchar(255))
81 BEGIN
82     DECLARE p_id int;
83     SELECT id INTO p_id FROM lab4.bankuser WHERE username=p_username AND
valid=true;
84     SELECT isadmin FROM lab4.bankidentity WHERE id=p_id;
85 END$$
86 DELIMITER ;
87
88 GRANT select ON lab4.bankuser TO 'admin'@'localhost';
89 GRANT select(id), select(currency), update(currency) ON lab4.bankcurrency TO
'admin'@'localhost';
90 GRANT select ON lab4.bankidentity TO 'admin'@'localhost';
91 GRANT EXECUTE ON PROCEDURE lab4.queryalluser TO 'admin'@'localhost';
92 GRANT EXECUTE ON PROCEDURE lab4.changecurrencybyusername TO 'admin'@'localhost';
93 GRANT EXECUTE ON PROCEDURE lab4.querysingleusercurrency TO 'admin'@'localhost';
94
95 GRANT select ON lab4.bankuser TO 'ziyang'@'localhost';
96 GRANT select ON lab4.bankcurrency TO 'ziyang'@'localhost';
97 GRANT select ON lab4.bankidentity TO 'ziyang'@'localhost';
98 GRANT EXECUTE ON PROCEDURE lab4.querysingleusercurrency TO 'ziyang'@'localhost';
99
100 GRANT select ON lab4.bankuser TO 'exp'@'localhost';
101 GRANT select ON lab4.bankcurrency TO 'exp'@'localhost';
102 GRANT select ON lab4.bankidentity TO 'exp'@'localhost';
103 GRANT EXECUTE ON PROCEDURE lab4.querysingleusercurrency TO 'exp'@'localhost';
104
105 FLUSH PRIVILEGES;

```