# 中山大学

# 本科生实验报告

|  |
| --- |
| 题目：图书馆管理系统 |
| 学院：数据科学与计算机学院 |
| 专业：16级计算机9班 |
| 姓名：李新锐 |
| 学号：15323032 |
| 日期：2017/6/10 |

## 一.实验目的

本实验面向C++语言的初学者。

主要让实验者熟悉面向对象的编程思想以及类的使用。

## 二. 实验环境

**2.1 编程语言和开发工具**

编程语言： C++、QML

开发工具： QT5.8/MSVC2015

**2.2 编码规范**

采用LLVM规范

## 实验内容

实现图书馆管理系统。

## 分析与设计

简要描述程序设计的过程，包括设计思路，设计要点及特色；程序的不足与改进等。要求画出程序的简单流程图。

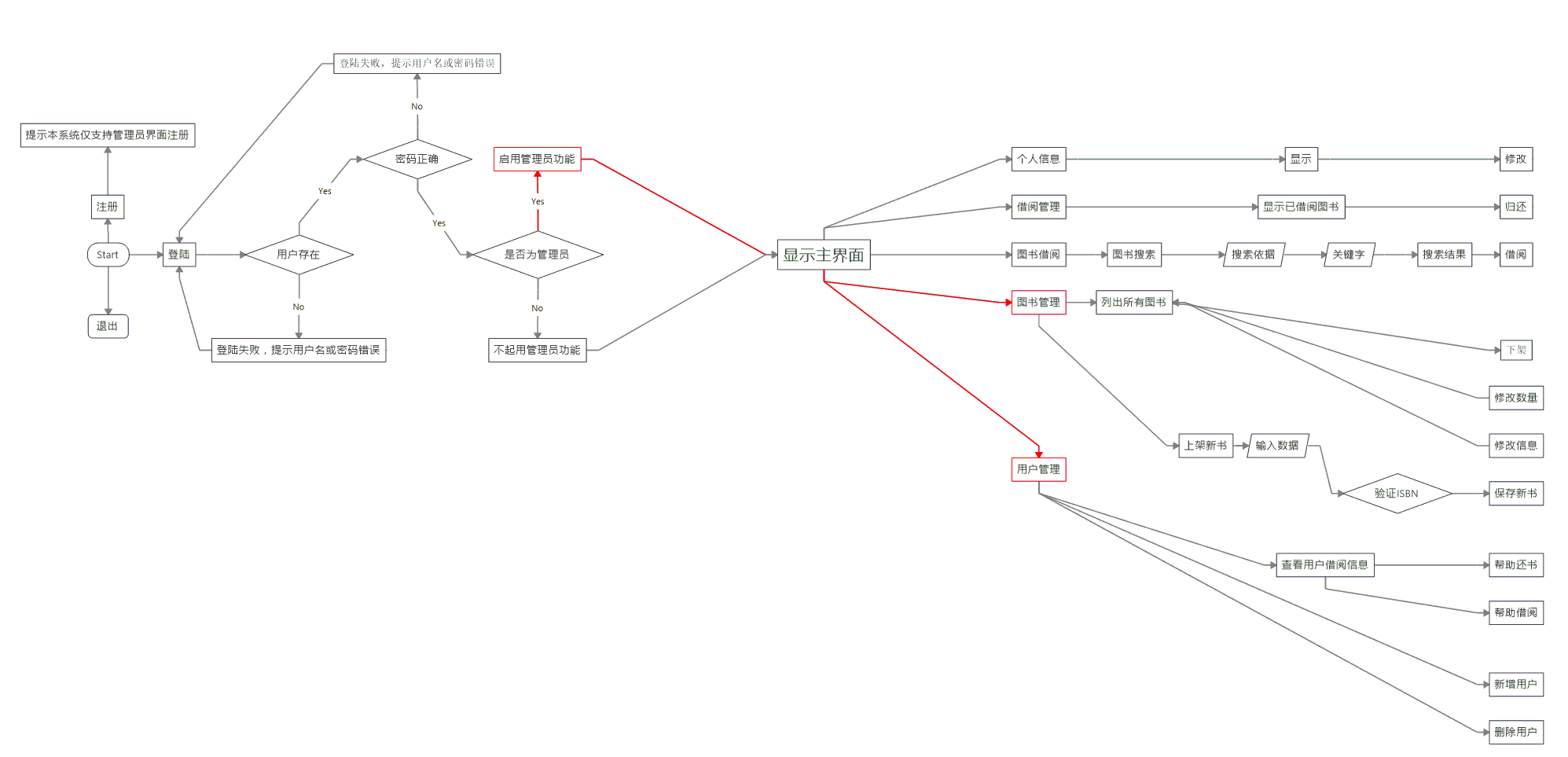
## 4.1 需求分析：

1. 用户需要了解自己借了那些书。对于一本书籍，也需要能够列出所有借阅者。此外对书或读者操作时（如借书还书），与其关联的读者或书可能需要做出响应。
2. 用户普遍希望有更加友好的用户界面。目前用户普遍拥有多种类型，不同分辨率，dpi的设备，在这些设备上用户希望拥有同样的良好体验。
3. 用户的输入可能出错，需要验证。
4. 管理员具有特权操作，需要进行权限区分。用户信息也需要保护，必须考虑密码安全。
5. 项目规模较大，需要合适的架构。
6. 数据需要持久化储存。
7. 程序需要面对大量数据稳定运行，因此要考虑内存资源安全。
8. 书籍数量变大后，用户不可能一本本寻找。

## 4.2 实现思路

原则：由类的用户代码，反推类的接口；由类的接口，反推类的实现。

因此首先考虑了界面层的设计，如下图所示：

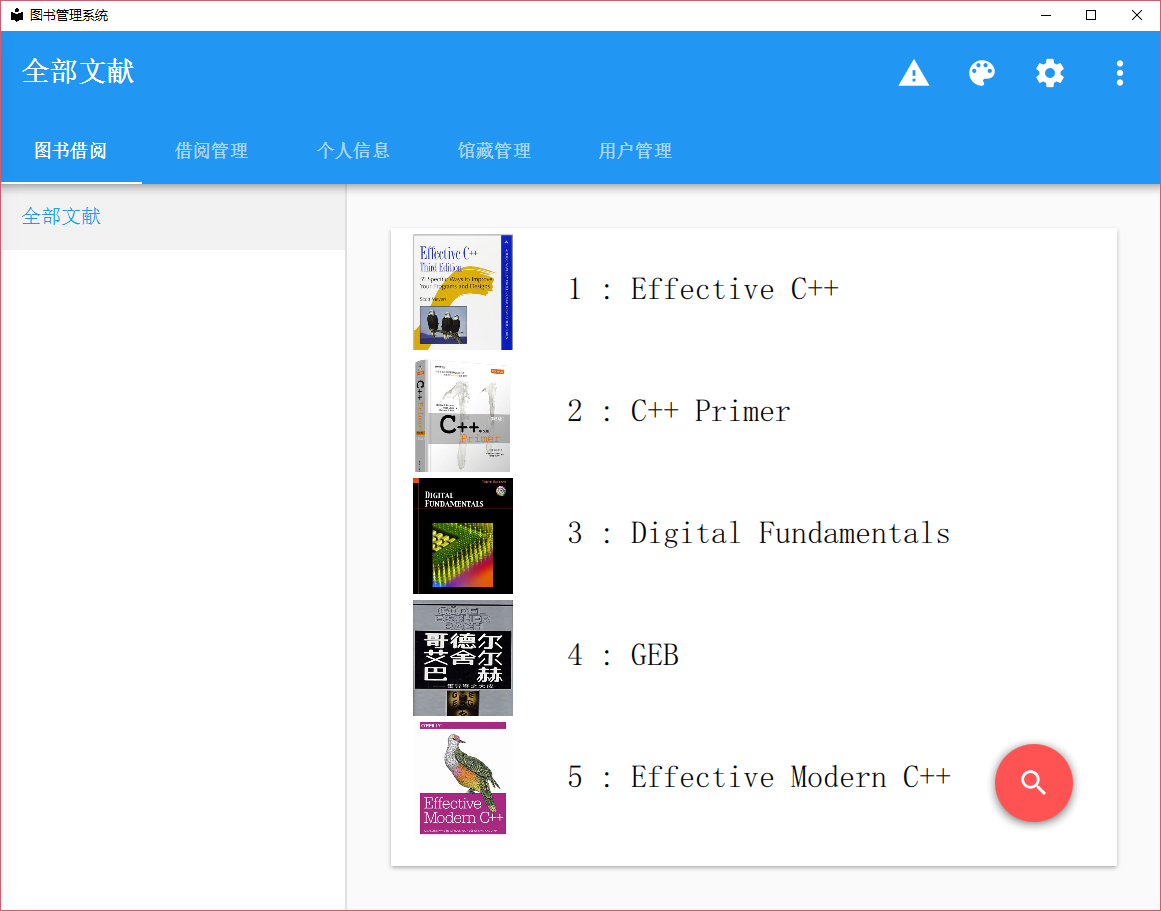


之后，逐一考虑上述需求的解决方案：

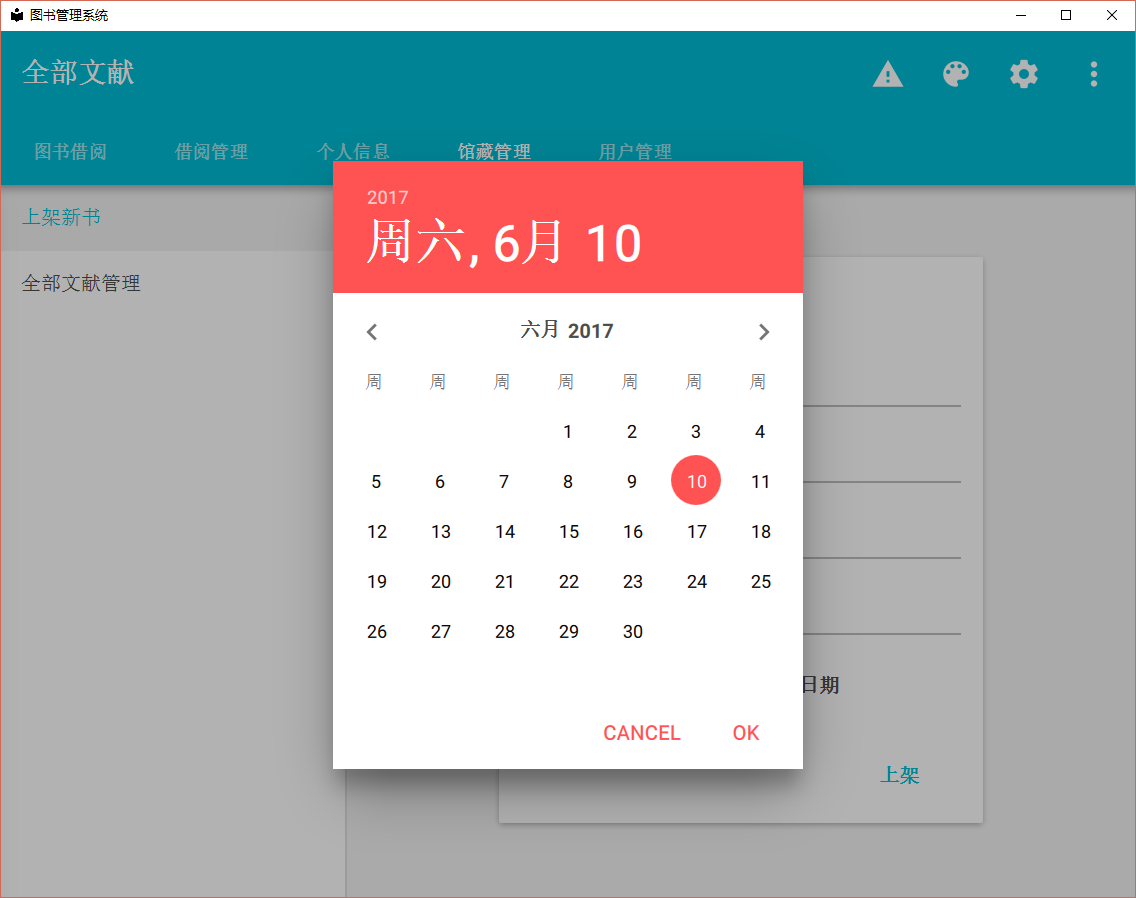
1. 可以采用关联模式，让Book类知道User类的实现，而User类不知道Book类的实现，仅知道Book类的前置声明。确保不发生两个类相互include的情况下，让Book类/User类分别保管一个内容是User\*/Book\*的list容器，用以记录哪些用户借了该书/该用户借了那些书。Book类中借书还书函数不仅修改Book的数据（如存量改变），还通过指针调用对应User中的对应函数。在Book类中实现自定义的拷贝构造函数和析构函数，确保在被拷贝，析构删除时，与之相关的User做出响应。（参考C++ Primer ed.5 P520）。
2. 最初考虑使用Qt做图形界面，发现界面仍不够美观。后改用QML，这是一门面向原型的语言，具有与强大的C++交互能力，因此可用用来编写界面层。

界面设计采用谷歌Material Design设计风格。具有切换主题功能。所有尺寸均采用设备独立像素表示，控件布局采用锚（anchor）表示，因此在不同的dpi具有同样良好的显示效果，特别是兼容高dpi。

举管理员登陆后的界面为例，（此时为Windows设置了150%的缩放，dpi144）



1. 尽可能使用选项菜单，日期选择器等控件代替用户直接输入文本，最小化用户输入的可能性，减少错误。在必须输入文本的输入框处，使用正则表达式等方法，设置了输入合法性检查。如下图中上架新书时，新书出版日期使用日期选择器选择，并阻止了不合法的ISBN输入：



1. 程序区分管理员账户和普通账户，与账户密码、用户权限相关的记录与普通记录（用户姓名，书籍信息等）分两个数据库文件分别储存。密码采用MD5加密，在数据库中以密文储存，即使被拖库也不会被直接获取。
2. 项目整体采用三层架构，分为界面层，业务逻辑层，数据访问层。每一层对高层的存在不知情，而高层只可以调用低一层提供的公有接口，对底层的实现不知情。本项目中界面层使用QML实现，逻辑层使用C++实现，数据访问层通过C++调用sqlite实现。

具体到每一层又使用了不同的设计，界面层使用了Model-View-Delegate架构，逻辑层的UserModel传入做为Model，同时这个类也是界面层和逻辑层的唯一接口类，界面层对逻辑层的操作均由该类发起。业务逻辑层中，User类聚合到UserModel上。UserModel的初始化通过数据访问层提供的哈希表接口获取二进制数据，再进行搜索把数据转换回来，使数据访问层不需要知道逻辑层任何类的表示形式。数据访问层中，由于有记录普通数据的数据库和记录密码信息的安全数据库两种，故采用了工厂方法模式更好地封装了创建数据库对象的细节，同时提高了可扩展性。

1. 数据的持久化储存使用了轻量级SQL数据库sqlite。每个数据库分别包含在磁盘上的一文件中，具有高迁移性。同时该程序是单用户的，不需要考虑并发问题，因此sqlite是很好的选择。
2. 本程序三层架构每一层采用了不同方式保证内存资源安全：

数据访问层所有出现资源分配的地方，均采用在资源产生的那一时刻立即分配给容器(shared\_ptr)进行管理的方法，并且确保了所有该层代码不出现裸指针。(参考Effective C++ Term 38)

逻辑层各类均继承了QObject类，对于这些类Qt采用父子对象管理机制，实现了自动垃圾回收。

界面层中QML拥有类似JavaScript的内存回收机制。

## 4.3 OOP三个特征的体现

**1. 封装**

本程序中封装体现在三层架构层与层之间的封装和同层之间不同类之间的封装。

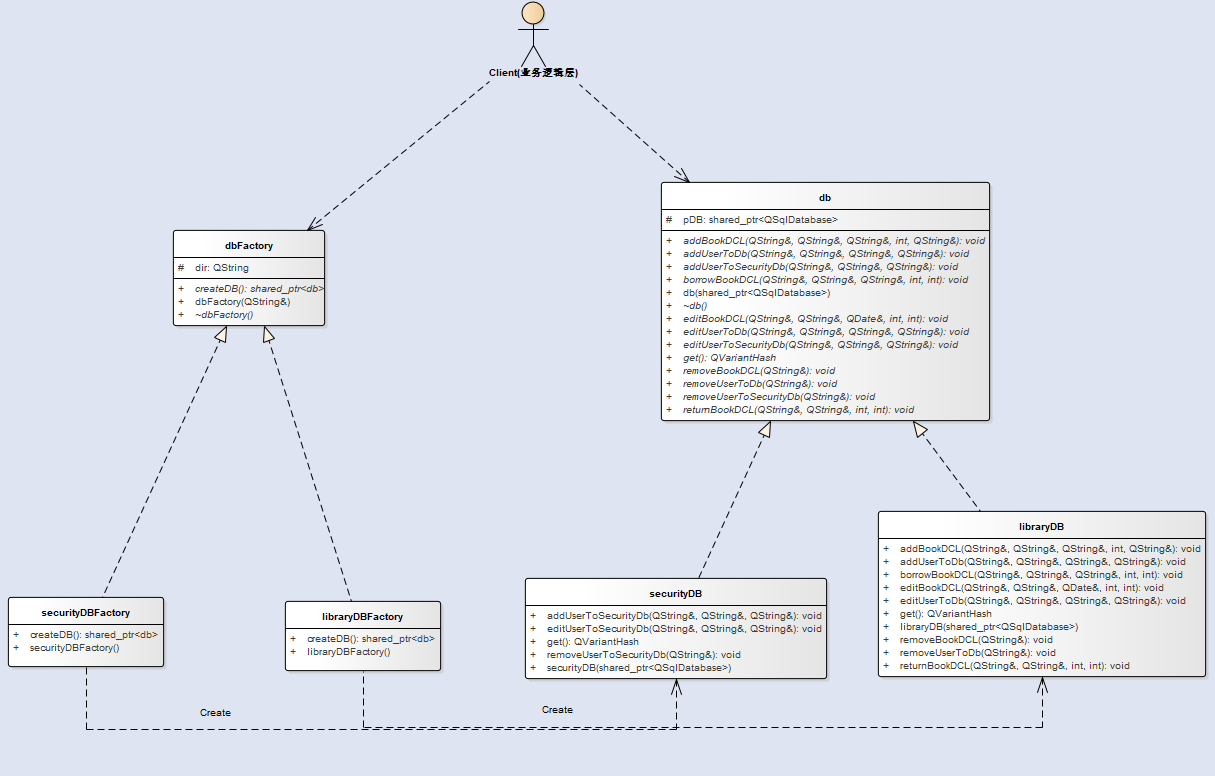
层与层之间的封装：界面层，业务逻辑层，数据访问层这三层每一层对高层的存在不知情的同时，对高层封装了自己的实现细节，只允许高层调用它提供的公有接口。

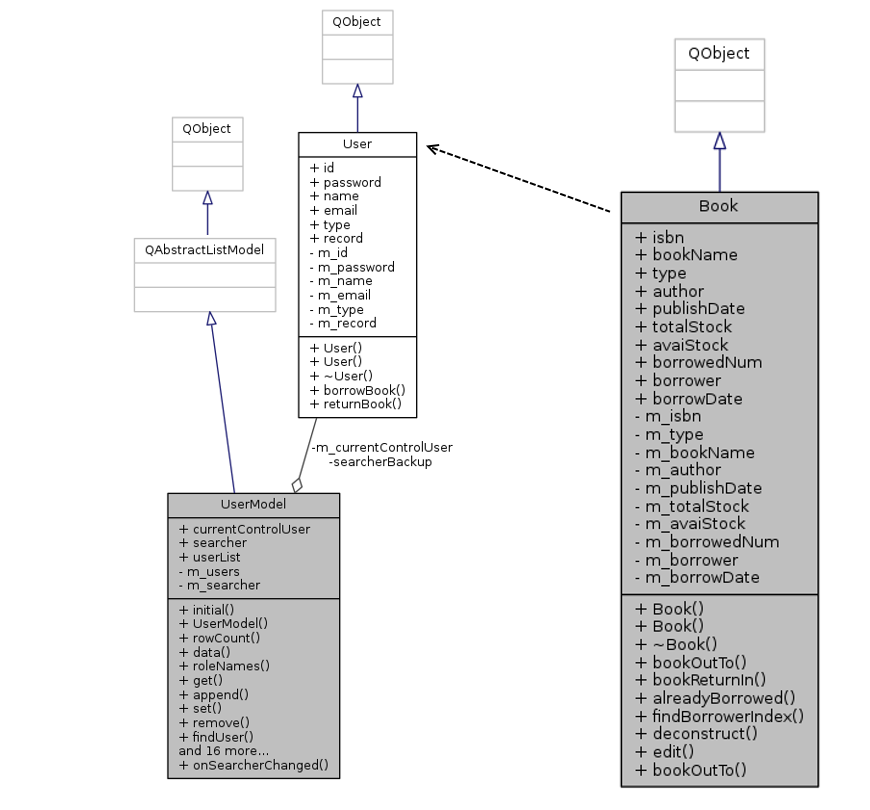
逻辑层的UserModel类是界面层和逻辑层的唯一接口，界面层通过该类读取要要展示的数据，同时由该类发起对逻辑层增删查改的操作。而数据访问层的db类则是数据访问层与逻辑层的唯一接口，提供get方法返回无格式的哈希表数据用于逻辑层读取数据初始化UserModel，并提供接口用于持久化逻辑层对用户、书籍的修改操作。

同层之间：数据访问层中数据库的工厂类提供createDB接口，类封装了连接数据库文件进而产生db产品的具体细节。逻辑层中，User类提供了必要的增删查改函数给UserModel类，封装了具体实现细节。UserModel类又重新封装接口给界面层。

**2.继承与多态**

两张UML图分别展示了数据层/逻辑层各个类之间继承与多态的关系（可放大查看，解释见下一页）

****

****

如第一张UML图所示，db类接口充当抽象产品，其子类libraryDB和securityDB继承db类，利用多态特征重写(override)了基类的get等虚函数，是对逻辑层的接口的具体实现，与基类是实现关系；类似的dbFactory接口充当抽象工厂，其子类libraryDBFactory和securityDBFactory继承它充当具体工厂，并利用多态实现各自的createDB函数，这个函数用于创建db。

如第二排中两张UML图所示：

UserModel类继承QAbstractModel类，利用多态重写了该类的抽象的append，rowCount等函数，这使UserModel能在Model-View-Delegate架构的界面层中充当Model角色。

三个类均继承QObject类，因此具有上一节提过的自动垃圾回收机制等功能。

UserModel与User之间是聚合关系，Book类单向依赖User类（借书还书时通过指针调用对应User中的对应函数）

## 4.4 数据设计，类设计，函数设计

该部分详尽内容较多，请点击[该链接](https://d.docs.live.net/790ab8a494d9c00b/Documents/大学课程/程设二/doc/index.html)或直接打开doc目录下的index.html进行查看。

其中重点内容包括：

数据设计上，QVariantHash db::get()返回的Hash表代表了整个从内存中读取的一个数据库文件，逻辑层在获取该表后可以从中初始化UserModel，是数据设计上的重点。

之所以采用这个类型，是因为一个数据库具有多个名称不同的tables，每个table有多条记录，一个记录中一个column对应一个值。使用嵌套Hash表能很好地表示这个关系。

这个Hash表的类型QVariantHash 是QHash<QString, QVariant>的typedef，代表一个table名称与这个table的对应。QVariant类似std::variant，它保有其一个可选类型之一的值，要么不保有值，在这里它实际保有的是QVariantList，它是QList<QVariant>的typedef。而这个list的QVariant元素实际保有的类型又是QVariantHash，即QHash<QString, QVariant>，QVariant是QString，向这个QVariantHash中插入每个column的值，就可以代表一条记录。

类设计上，贯彻面向对象的封装，继承，多态原则，在上一节已经叙述过。此外，由于需要有展示全部书籍的需求，故创造了了一个virutalUser\_AllBooks用户，它拥有全部图书。在查看全部书籍时UserModel类将向界面层报告当前用户为virutalUser\_AllBooks，这样就保持了整个设计行为上的一致性。

函数设计上，由于C++的类向QML导入时，每个成员变量都要实现读、写、信号三个函数，并且有严格的格式要求，重复书写较为麻烦，因此采用了宏编程的方法，用以下宏实现批量产生所需的三个函数。

#define DEF\_PROPERTY\_IMPL(type, propertyName, PropertyName) \

type propertyName() const { return m\_##propertyName; } \

void set##PropertyName(const type &\_##propertyName) { \

m\_##propertyName = \_##propertyName; \

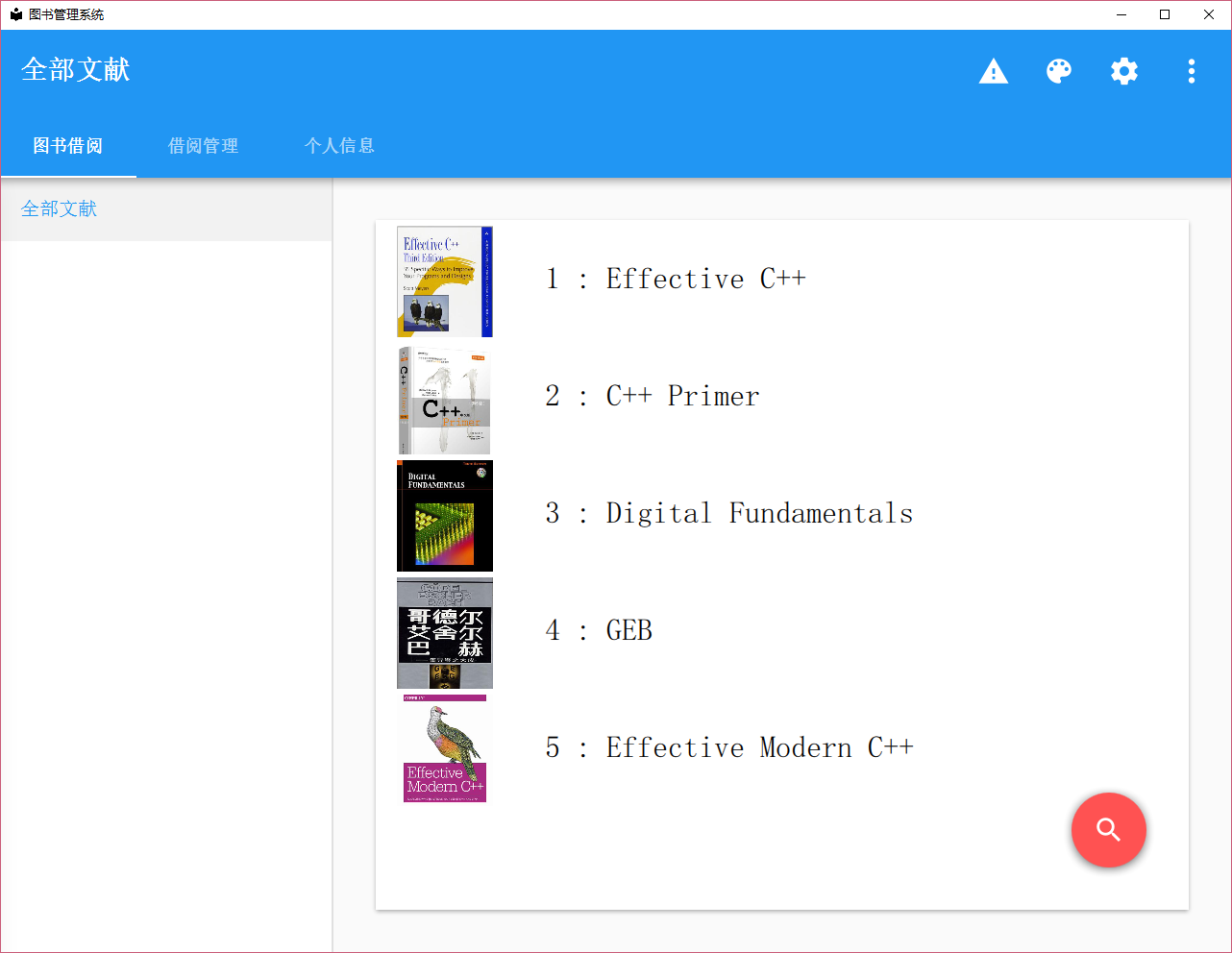
emit propertyName##Changed(); \

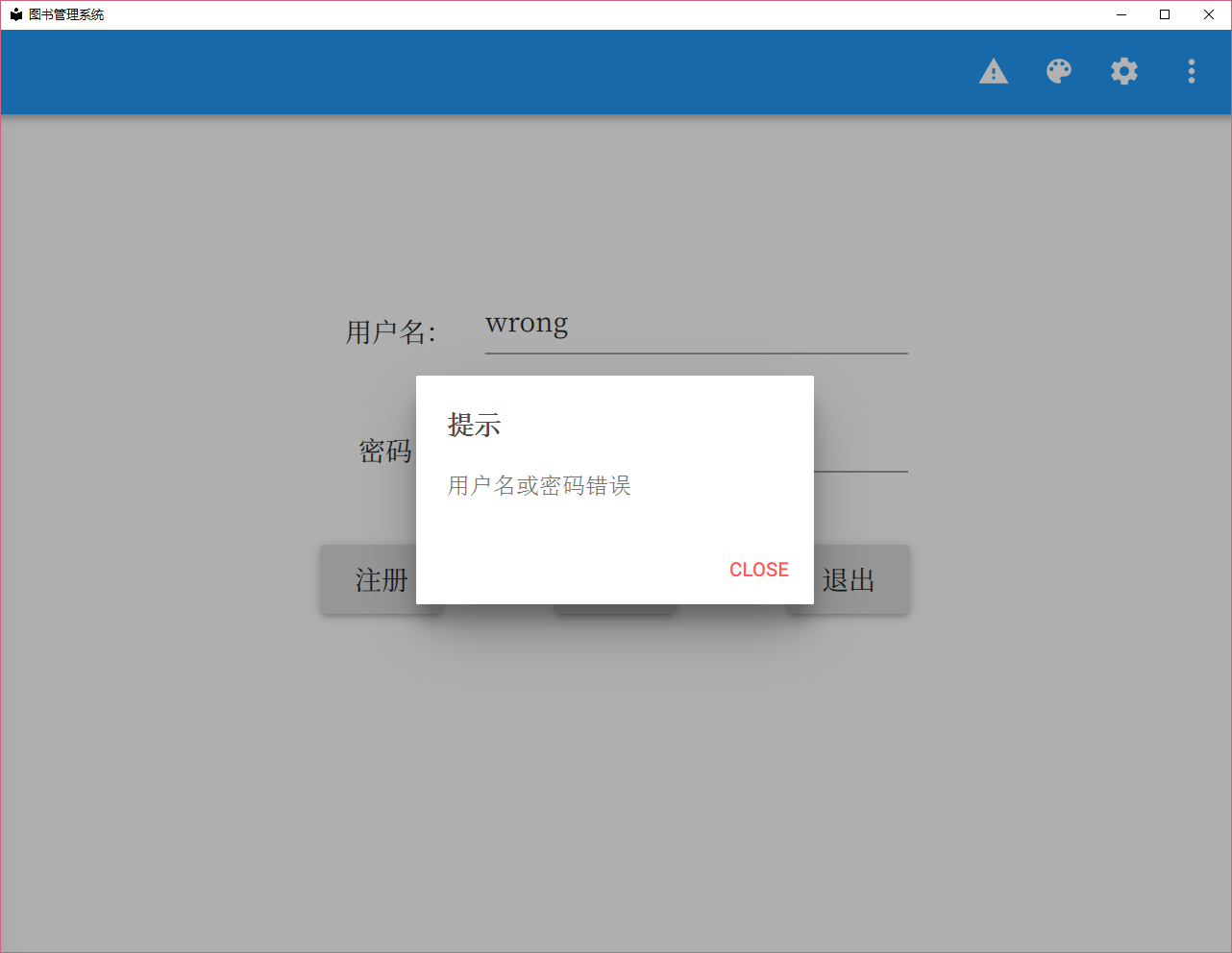
}

# 实验结果

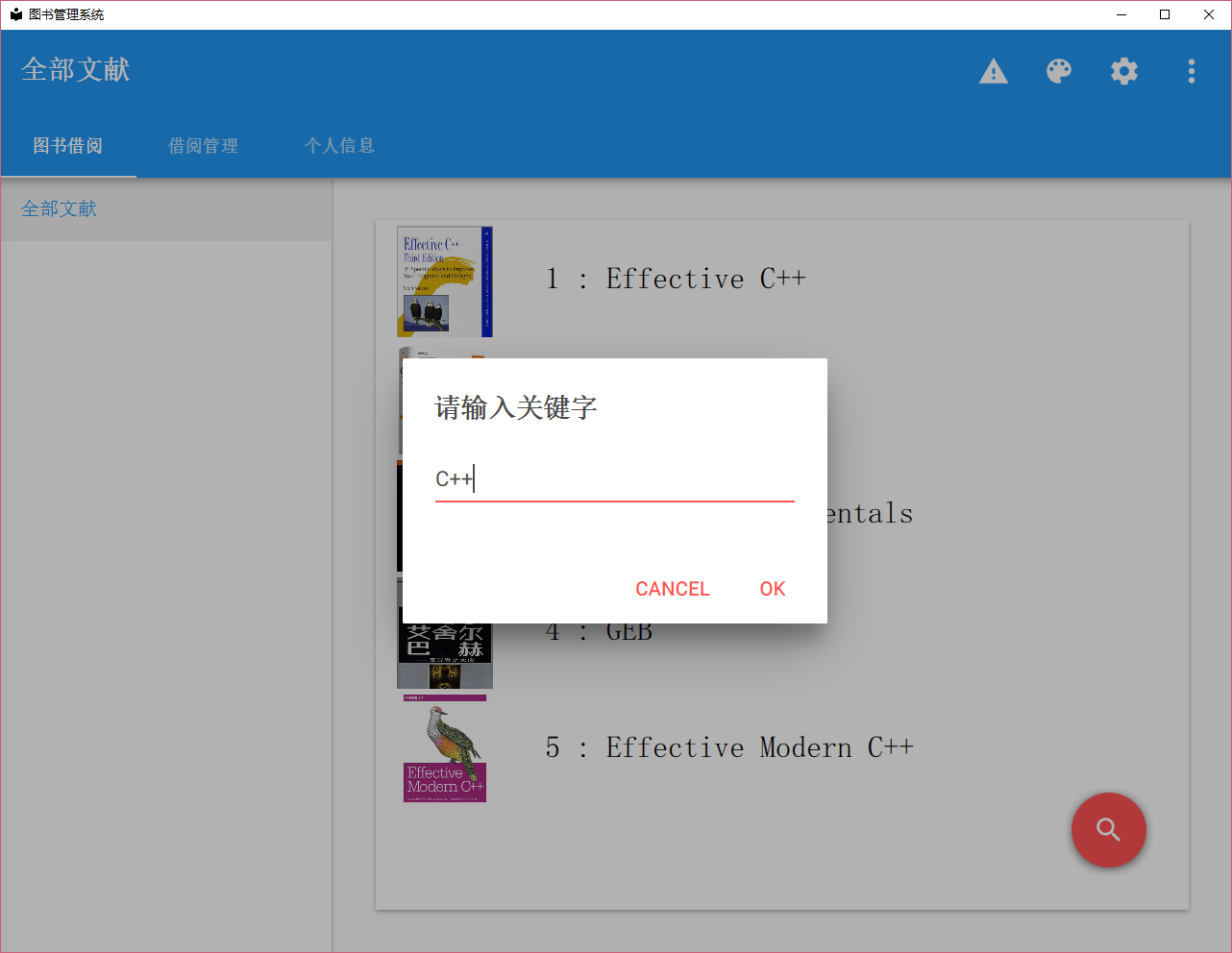
1. 登录界面

以普通用户身份登陆

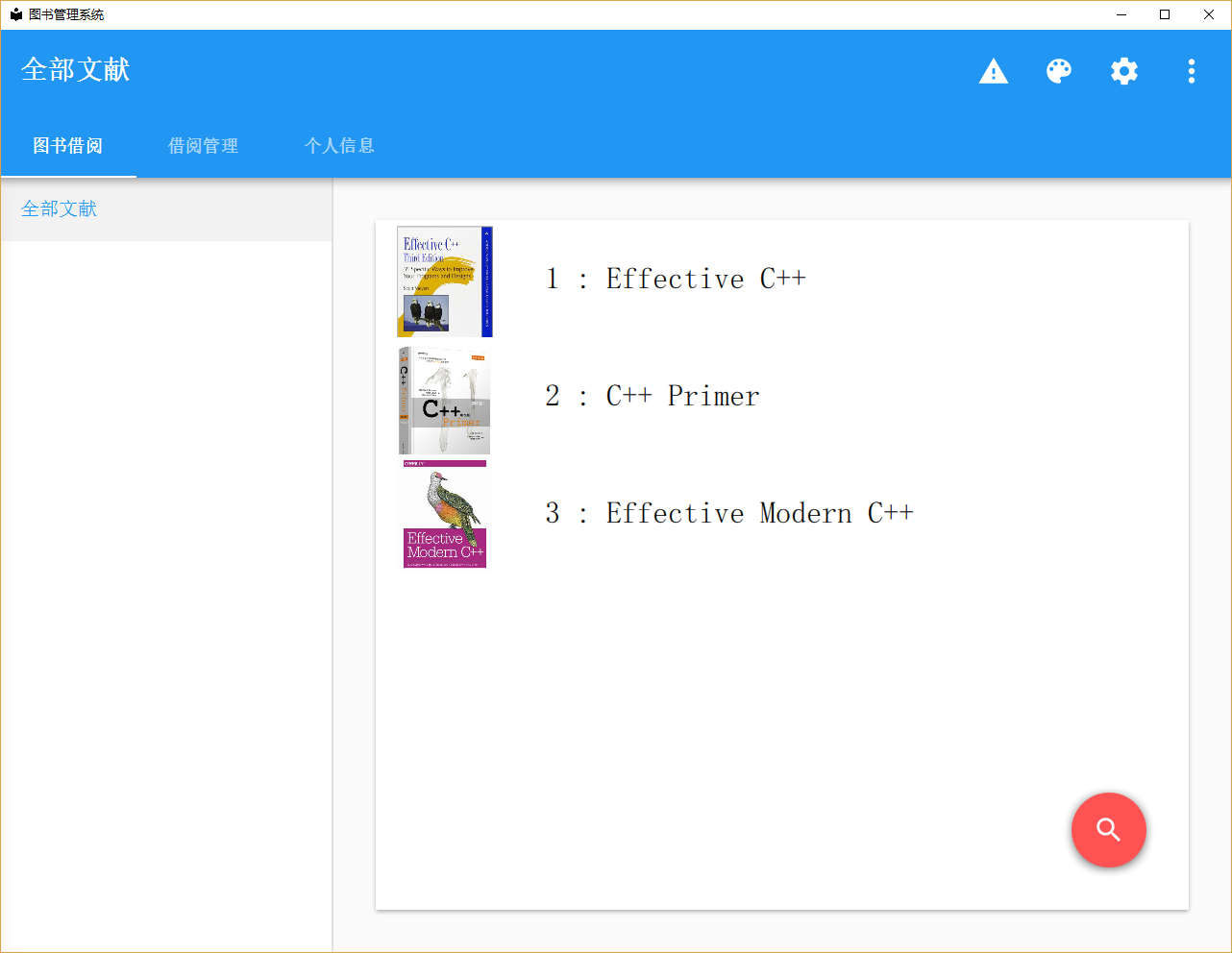


如果输错用户名或密码

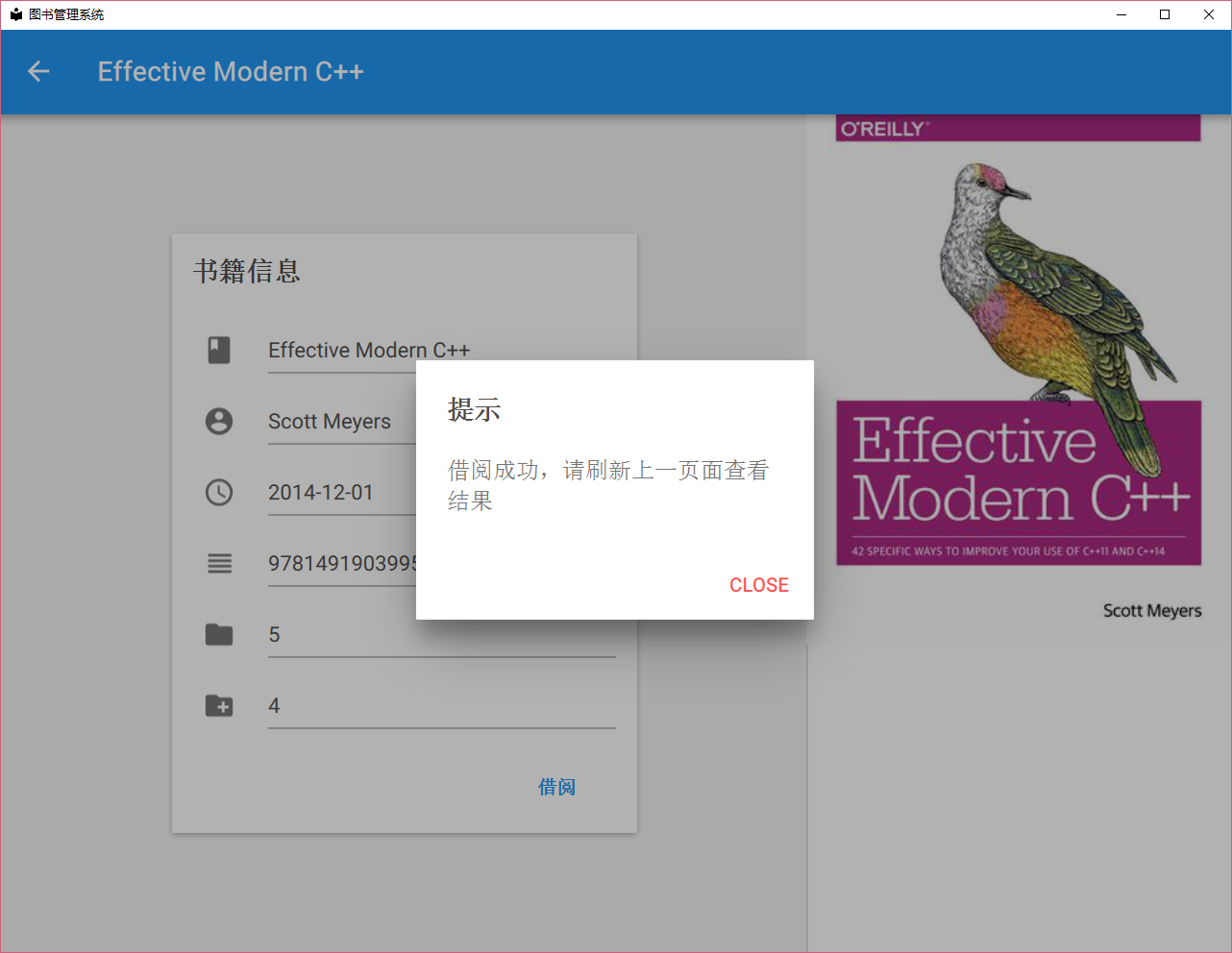
1. 搜索界面

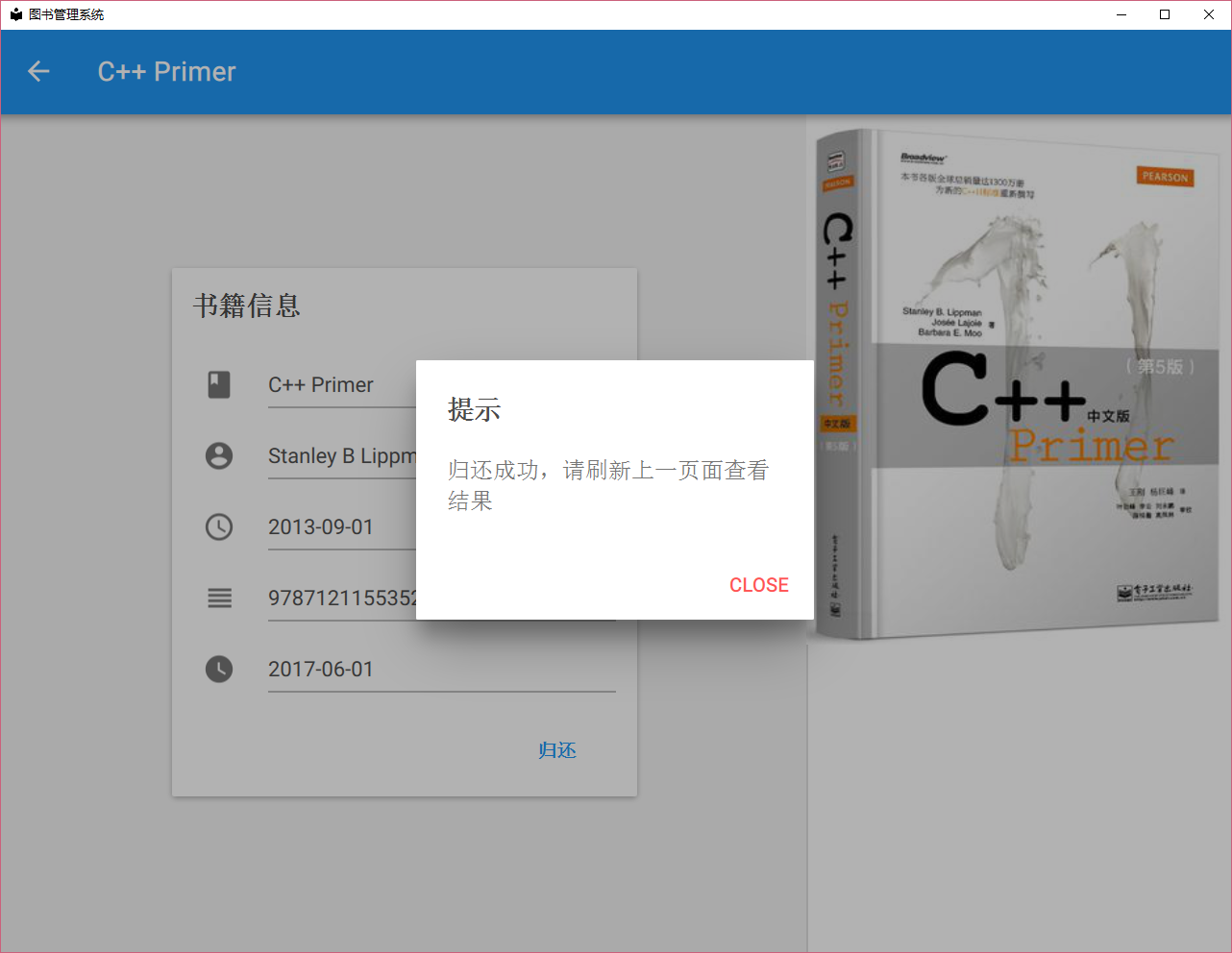
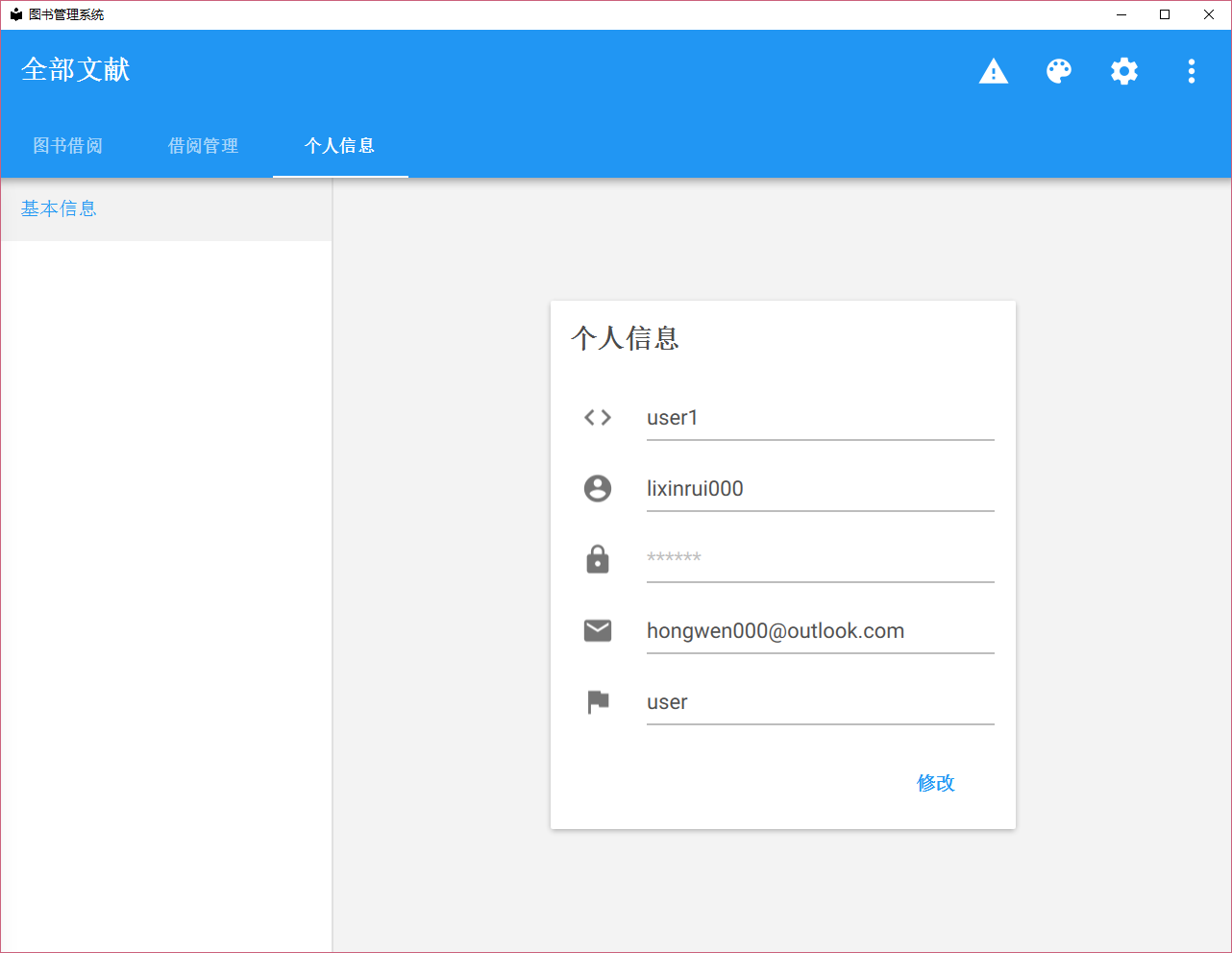
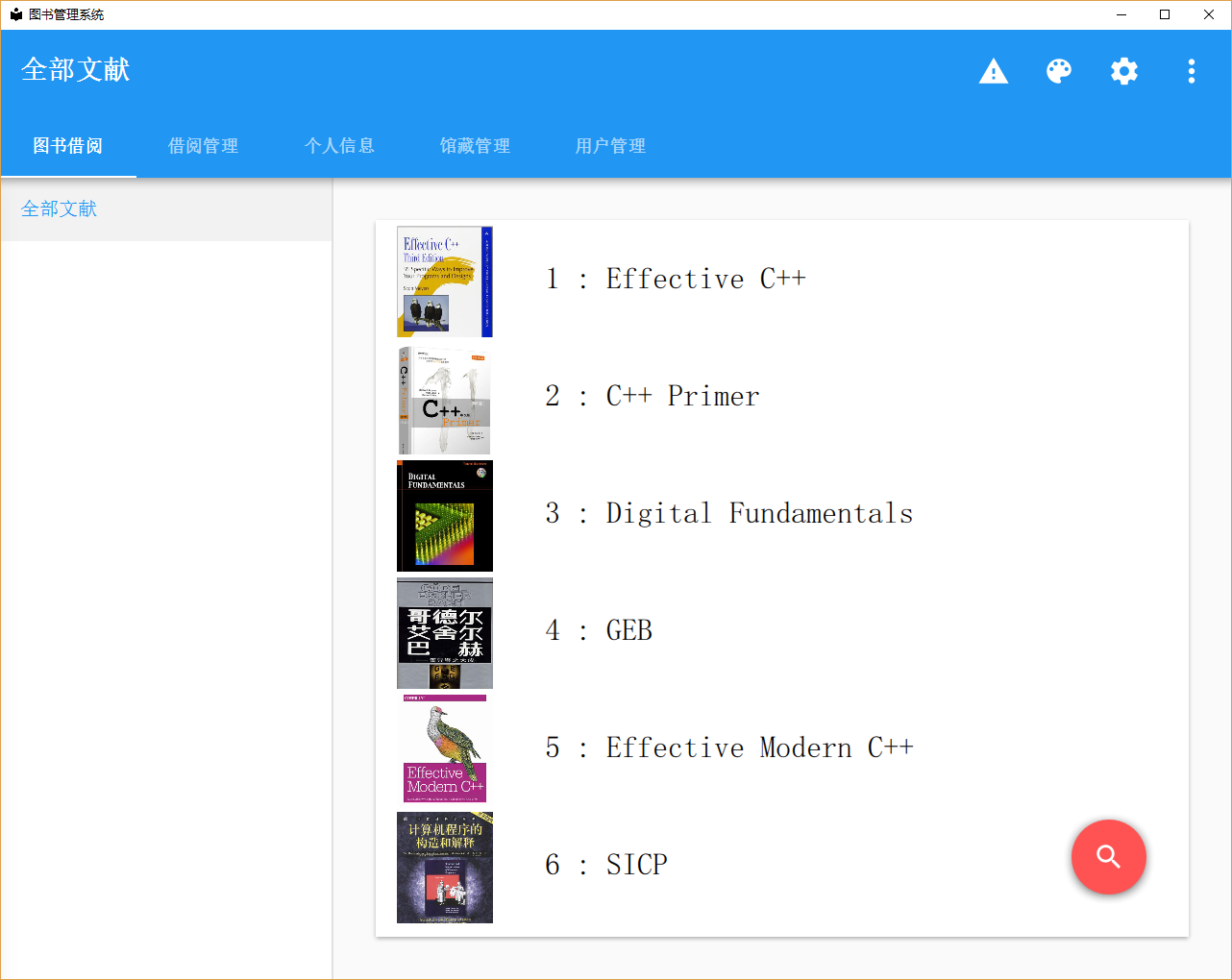
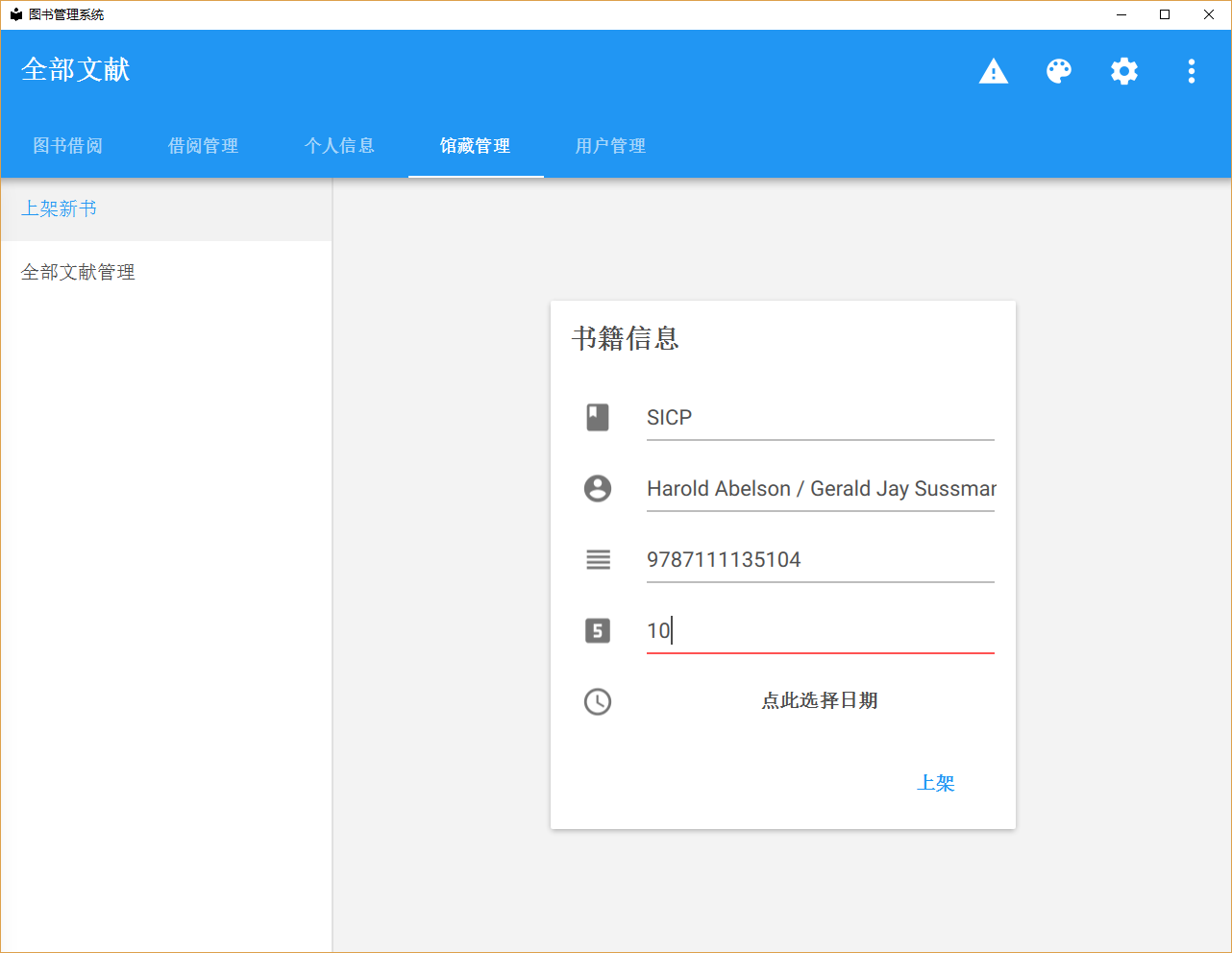
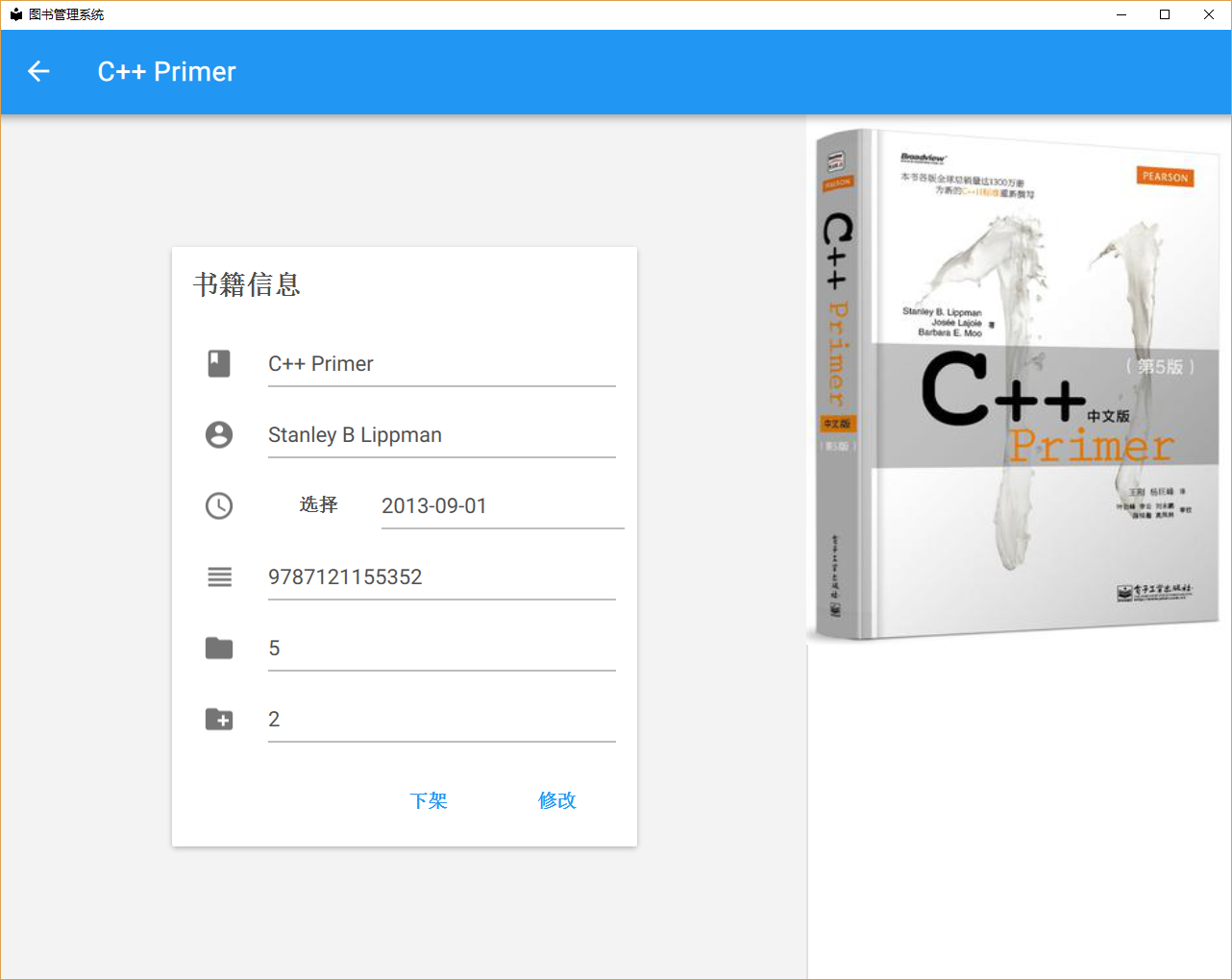
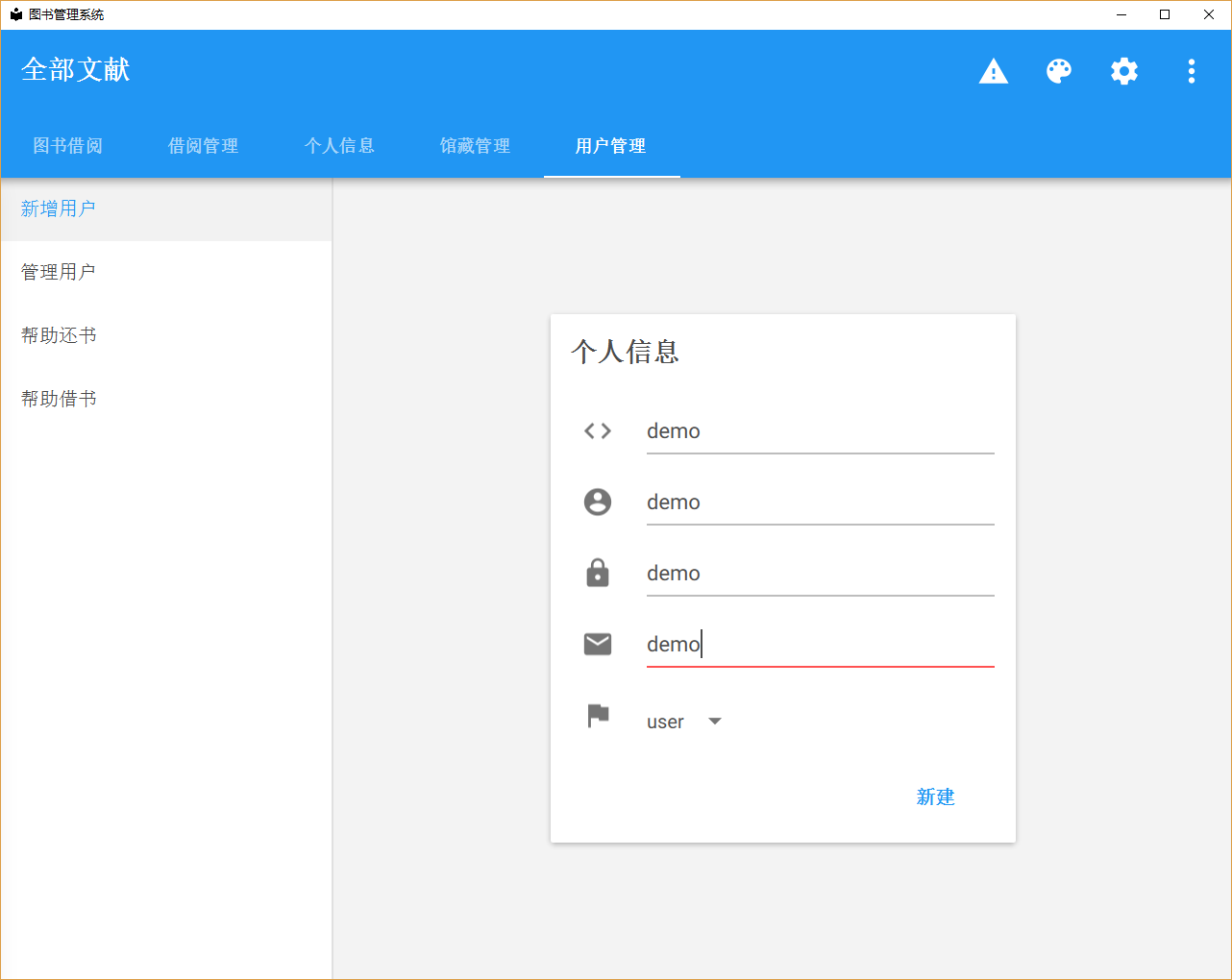
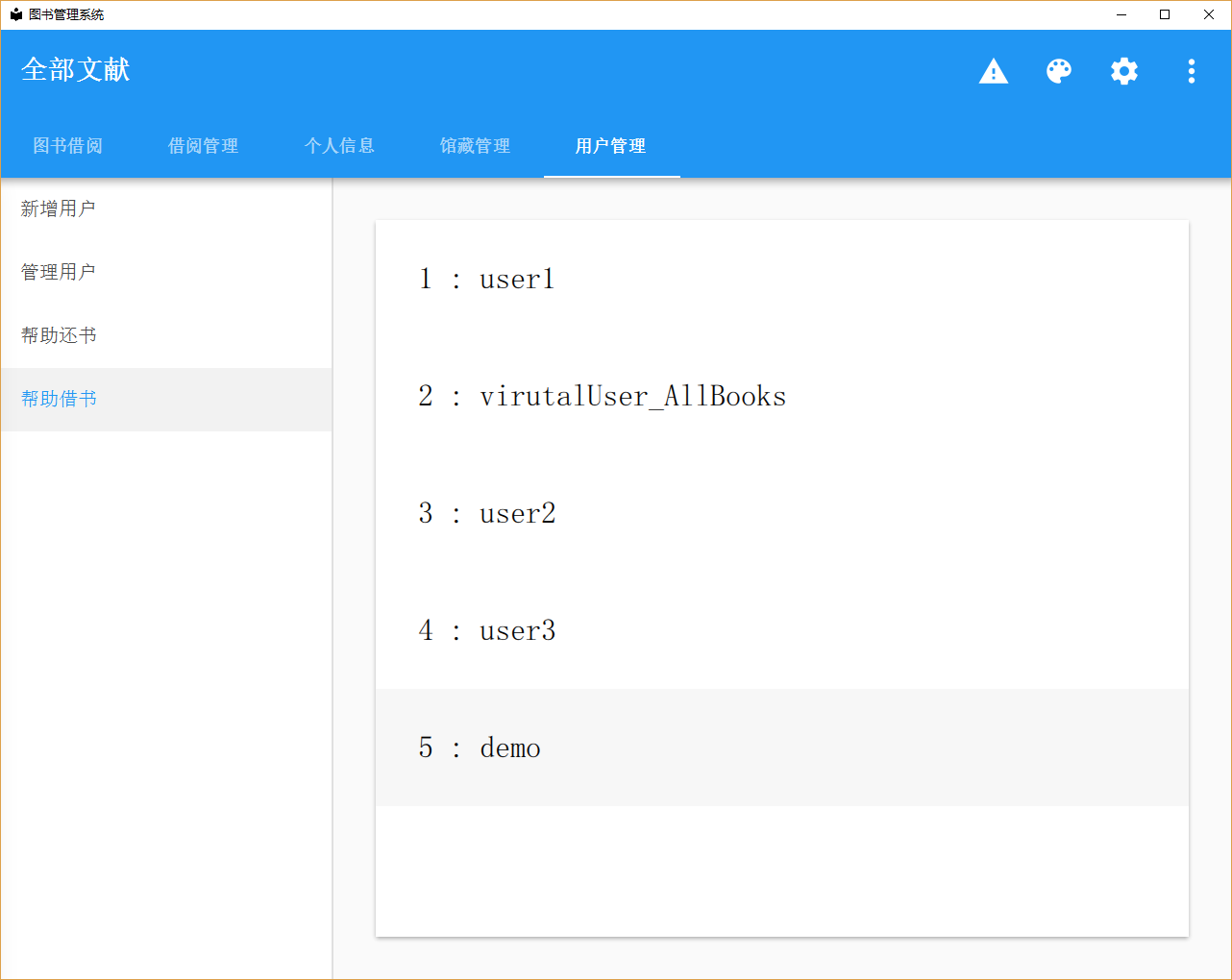


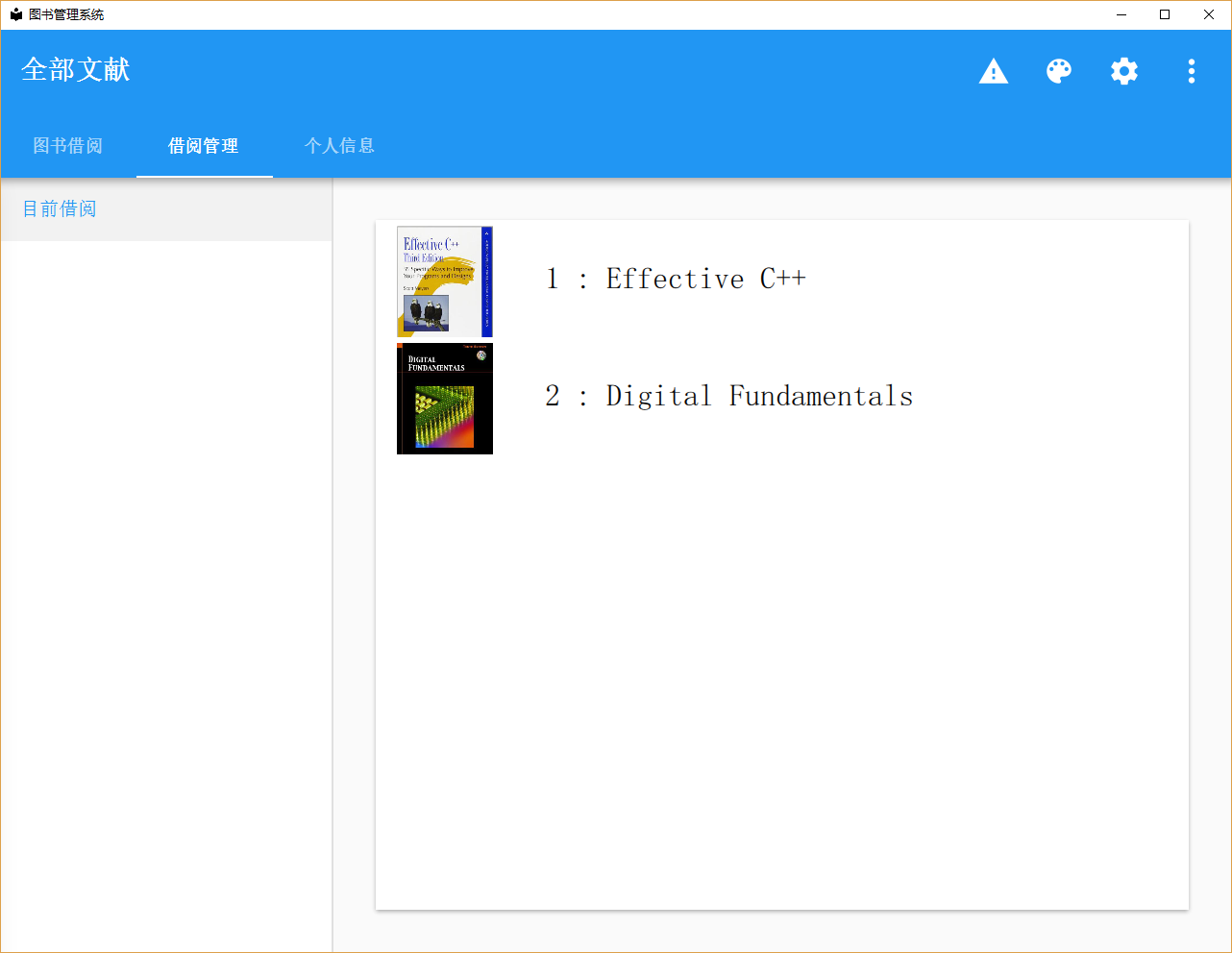
查看结果



1. 借书



1. 还书
2. 查看/修改个人信息
3. 以管理员身份登陆
4. 新增书籍
5. 修改/下架书籍
6. 新增用户
7. 帮助借书

重新打开程序，登陆该用户，查看结果

1. 使用终端打开数据库验证数据已经保存：

db λ sqlite3 library.db

sqlite> .schema

CREATE TABLE user (id VARCHAR PRIMARY KEY,

name VARCHAR,

email VARCHAR,

type VARCHAR);

CREATE TABLE book (isbn VARCHAR PRIMARY KEY,

bookName VARCHAR,

author VARCHAR,

publishDate VARCHAR,

totalStock INTEGER,

avaiStock INTEGER,

borrowedNum INTEGER);

CREATE TABLE trans (book VARCHAR,

user VARCHAR,

borrowDate VARCHAR);

sqlite> select \* from user;

user1|lixinrui000|hongwen000@outlook.com|user

virutalUser\_AllBooks|||admin

user2|lixinrui Admin|hongwen000@outlook.com|admin

user3|user3|hhh@hhh.com|admin

demo|demo|demo|user

sqlite> select \* from book;

9787121029097|Effective C++|Scott Meyers|2006-07-01|5|1|9

9780132359238|Digital Fundamentals|Thomas L Floyd|2006-06-01|5|1|9

9787100013239|GEB|Douglas Richard Hofstadter|1997-05-01|5|3|7

9781491903995|Effective Modern C++|Scott Meyers|2014-12-01|5|3|7

9787111135104|SICP|Harold Abelson / Gerald Jay Sussman / Julie Sussman |2004-02-02|10|9|1

sqlite> select \* from trans;

9787121029097|virutalUser\_AllBooks|2006-07-01

9780132359238|virutalUser\_AllBooks|2006-07-01

9787100013239|virutalUser\_AllBooks|2006-07-01

9780132359238|user1|2017-06-01

9780132359238|user2|2017-06-01

9787121029097|user1|2017-07-03

9787100013239|user2|2017-07-05

9787121029097|user3|2017-07-07

9781491903995|virutalUser\_AllBooks|2017-07-07

9781491903995|user1|2017-07-11

9787111135104|virutalUser\_AllBooks|2017-07-11

9787121029097|demo|2017-07-11

9780132359238|demo|2017-07-11

db λ sqlite3 security.db

sqlite> .schema

CREATE TABLE security (id VARCHAR PRIMARY KEY,password VARCHAR, type VARCHAR);

sqlite> select \* from security;

user1|7ea2bd72bfc7dabdfecc0b5760ebcf52|user

user2|3d281d21c49d79f586af2cdc4419b18b|admin

virutalUser\_AllBooks|efe02f731a16df5853ed099773485933|admin

user3|ad9a9c4cfd2406cb966cd54d96194f6e|admin

demo|fe01ce2a7fbac8fafaed7c982a04e229|user

## 五、设计心得

过去三周，我从零开始，学习了Qt，QML，JavaScript，SQL，设计模式，成功写出了本Project，取得了较大收获。这段学习过程中遇到的困难，主要是面对一些全新的，又没有好的教程用来形成一个基础的知识体系，以及足够大的社区帮你在StackOverflow上问好各种问题，一个很简单的问题都会卡很久。例如学习QML时，官方的tutorial就是讲了一个简单的例子，并没有讲清每一句话的含义，大概是“你看，这里写个TextField{}，就会有个文本框”这种感觉。但，TextField及它里面的property、它的id作用域是什么呢，跨文件怎么办呢？（我大概有几天时间都在纠结各种"property xxx is not bound in this Context"）？

直到一周后去图书馆找参考书，在广图没有找到，还好学校图书馆有一本，帮助我建立了知识体系，对我起了很大帮助。