

浙江大学 数据库系统 课程

Lab5 图书管理系统



Anonymous

日期: 2024/03/25

2024-2025 春夏 学期

Table of Contents

1 实验目的	4
2 实验需求	4
2.1 基本数据对象	4
2.2 基本功能模块	4
2.3 数据库表设计	5
2.4 系统功能验证	6
3 实验环境	7
3.1 操作系统	7
3.2 IDE	7
3.3 前端	7
3.4 后端	8
3.5 数据库	8
4 模块设计	8
4.1 前端	8
4.1.1 Book.vue	9
4.1.2 Borrow.vue	11
4.1.3 Card.vue	13
4.2 后端	15
4.2.1 添加单本图书 (storeBook(Book book))	15
4.2.2 批量添加图书 (storeBook(List<Book> books))	15
4.2.3 增加图书库存 (incBookStock(int bookId, int deltaStock))	15
4.2.4 删除图书 (removeBook(int bookId))	16
4.2.5 修改图书信息 (modifyBookInfo(Book book))	16
4.2.6 借书 (borrowBook(Borrow borrow))	16
4.2.7 还书 (returnBook(Borrow borrow))	16
4.2.8 查询借阅历史 (showBorrowHistory(int cardId))	17
4.2.9 注册借书证 (registerCard(Card card))	17
4.2.10 删除借书证 (removeCard(int cardId))	17
4.2.11 修改借书证信息 (modifyCardInfo(Card card))	17
4.2.12 查询所有借书证 (showCards())	17
4.3 前后端 API	18
4.3.1 后端 API 设计	18
4.3.2 前端 API 设计	20
5 系统测试	23
5.1 正确性测试	23
5.2 功能性测试	25
5.2.1 图书管理页面	26
5.2.2 借书证管理页面	35

5.2.3 借书记录查询页面	38
6 问题与解决方案	38
6.1 前后端交互	38
6.1.1 出现的问题	38
6.1.2 解决方案	39
6.2 图书批量入库模块	39
6.2.1 出现的问题	39
6.2.2 解决方案	39
6.3 日期格式	41
6.3.1 出现的问题	41
6.3.2 解决方案	41
7 思考题	41
7.1 绘制该图书管理系统的 E-R 图	41
7.2 描述 SQL 注入攻击的原理(并简要举例)。在图书管理系统中，哪些模块可能会遭受 SQL 注入攻击？如何解决？	42
7.2.1 SQL 注入攻击的原理	42
7.2.2 易受攻击的板块	42
7.2.3 解决方案	42
7.3 在 InnoDB 的默认隔离级别(RR, Repeated Read)下，当出现并发访问时，如何保证借书结果的正确性？	43
7.3.1 InnoDB RR	43
7.3.2 对应策略	43
8 实验心得	44

1 实验目的

设计并实现一个精简的图书管理程序，具有图书入库、查询、借书、还书、借书证管理等功能。

- 完成一个基于 MySQL 的精简图书管理程序，具备较好的可扩展性、鲁棒性和安全性，并且在高并发场景下仍能正确运行。
- 完成类 LibraryManagementSystemImpl 中各功能模块的函数，并通过所有测试样例。同时，使用提供的前端框架，正确完成图书管理系统的前端页面，使其成为一个用户能真正使用的图书管理系统。
- 提高系统编程能力，加深对数据库系统原理及应用的理解。

2 实验需求

2.1 基本数据对象

所有的基本数据对象都被定义在 entities 包中，以下是这些对象的基本信息，其它信息参考类中的注释。

对象名称	类名	包含属性
书	Book	书号, 类别, 书名, 出版社, 年份, 作者, 价格, 剩余库存
借书证	Card	卡号, 姓名, 单位, 身份(教师或学生)
借书记录	Borrow	卡号, 书号, 借书日期, 还书日期

2.2 基本功能模块

图书管理系统中所有应具备的功能模块都在接 LibraryManagementSystem 中被声明,具体实现位于类 LibraryManagementSystemImpl 中。

在完成所有的接口后，在主类 Main 中通过 LibraryManagementSystem 的一个实例来实现与用户的交互。这样设计的好处在于它满足了：

1. 封装性：对客户端(主类 Main)屏蔽了图书管理系统底层的具体实现
2. 可扩展性：可以很方便地对图书管理系统做不同的实现(例如再实现一个不依赖于数据库的基于内存的图书管理系统)
3. 可维护性：只需要在客户端中切换 LibraryManagementSystem 的实例即可使用不同实现方式的图书管理系统

以下是 LibraryManagementSystem 中声明的模块，有关各模块的详细注释和说明在接口 LibraryManagementSystem 中给出：

- ApiResult storeBook(Book book)：图书入库模块。向图书库中注册(添加)一本新书，并返回新书的书号。如果该书已经存在于图书库中，那么入库操作将失败。当且仅当书<

类别, 书名, 出版社, 年份, 作者>均相同时, 才认为两本书相同。请注意, book_id 作为自增列, 应该插入时由数据库生成。插入完成后, 需要根据数据库生成的 book_id 值去更新 book 对象里的 book_id。

- `ApiResult incBookStock(int bookId, int deltaStock)`: 图书增加库存模块。为图书库中的某一本书增加库存。其中库存增量 `deltaStock` 可正可负, 若为负数, 则需要保证最终库存是一个非负数。
- `ApiResult storeBook(List<Book> books)`: 图书批量入库模块。批量入库图书, 如果有一本书入库失败, 那么就需要回滚整个事务(即所有的书都不能被入库)。
- `ApiResult removeBook(int bookId)`: 图书删除模块。从图书库中删除一本书。如果还有人尚未归还这本书, 那么删除操作将失败。
- `ApiResult modifyBookInfo(Book book)`: 图书修改模块。修改已入库图书的基本信息, 该接口不能修改图书的书号和存量。
- `ApiResult queryBook(BookQueryConditions conditions)`: 图书查询模块。根据提供的查询条件查询符合条件的图书, 并按照指定排序方式排序。查询条件包括: 类别点查(精确查询), 书名点查(模糊查询), 出版社点查(模糊查询), 年份范围查, 作者点查(模糊查询), 价格范围差。如果两条记录排序条件的值相等, 则按 `book_id` 升序排序。
- `ApiResult borrowBook(Borrow borrow)`: 借书模块。根据给定的书号、卡号和借书时间添加一条借书记录, 然后更新库存。若用户此前已经借过这本书但尚未归还, 那么借书操作将失败。
- `ApiResult returnBook(Borrow borrow)`: 还书模块。根据给定的书号、卡号和还书时间, 查询对应的借书记录, 并补充归还时间, 然后更新库存。
- `ApiResult showBorrowHistory(int cardId)`: 借书记录查询模块。查询某个用户的借书记录, 按照借书时间递减、书号递增的方式排序。
- `ApiResult registerCard(Card card)`: 借书证注册模块。注册一个借书证, 若借书证已经存在, 则该操作将失败。当且仅当<姓名, 单位, 身份>均相同时, 才认为两张借书证相同。
- `ApiResult removeCard(int cardId)`: 删除借书证模块。如果该借书证还有未归还的图书, 那么删除操作将失败。
- `ApiResult showCards()`: 借书证查询模块。列出所有的借书证。

2.3 数据库表设计

以下是该图书管理系统的数据表定义, 具体位于 `src/main/resources/db.sql` 中:

```
create table `book` (  
  `book_id` int not null auto_increment,  
  `category` varchar(63) not null,  
  `title` varchar(63) not null,  
  `press` varchar(63) not null,  
  `publish_year` int not null,  
  `author` varchar(63) not null,
```

```

`price` decimal(7, 2) not null default 0.00,
`stock` int not null default 0,
primary key (`book_id`),
unique (`category`, `press`, `author`, `title`, `publish_year`)
);

create table `card` (
  `card_id` int not null auto_increment,
  `name` varchar(63) not null,
  `department` varchar(63) not null,
  `type` char(1) not null,
  primary key (`card_id`),
  unique (`department`, `type`, `name`),
  check (`type` in ('T', 'S'))
);

create table `borrow` (
  `card_id` int not null,
  `book_id` int not null,
  `borrow_time` bigint not null,
  `return_time` bigint not null default 0,
  primary key (`card_id`, `book_id`, `borrow_time`),
  foreign key (`card_id`) references `card`(`card_id`) on delete cascade on update cascade,
  foreign key (`book_id`) references `book`(`book_id`) on delete cascade on update cascade
);

```

2.4 系统功能验证

系统功能验证测试分为功能性测试和正确性测试。

- 正确性测试通过测试用例进行评判, 以验收时通过的测试用例数量占总测试用例数量的百分比来评定正确性测试部分的得分。
- 当实现了前端, 图书管理系统表现为一个完整可使用的程序时, 进行功能性测试。功能性测试通过验收时随机运行模拟场景的结果进行评判, 以软件使用时的交互友好程度、效率、正确性等指标来评定功能性测试部分的得分。

对象名称	描述
图书入库	输入<书号, 类别, 书名, 出版社, 年份, 作者, 价格, 初始库存>, 入库一本新书 B
增加库存	将书 B 的库存增加到 X, 然后减少到 1
修改图书信息	随机抽取 N 个字段, 修改图书 B 的图书信息
批量入库	输入图书导入文件的路径 U, 然后从文件 U 中批量导入图书
添加借书证	输入<姓名, 单位, 身份>, 添加一张新的借书证 C

对象名称	描述
查询借书证	列出所有的借书证
借书	用借书证 C 借图书 B, 再借一次 B, 然后再借一本书 K
还书	用借书证 C 还掉刚刚借到的书 B
借书记录查询	查询 C 的借书记录
图书查询	从查询条件<类别点查(精确查询), 书名点查(模糊查询), 出版社点查(模糊查询), 年份范围查, 作者点查(模糊查询), 价格范围差>中随机选取 N 个条件, 并随机选取一个排序列和顺序

3 实验环境

使用 ZJUGit 上的代码作为框架, 对其进行调整和完善, 形成最终的图书管理系统。

3.1 操作系统

MacOS Macbook Air M4

- 芯片: Apple M4
- 内存: 24GB RAM
- CPU 核数: 10

3.2 IDE

使用 VSCode + Java + Maven 进行开发:

- VSCode: Visual Studio Code 是一个轻量级但功能强大的源代码编辑器, 支持多种编程语言和框架。通过安装 Java 扩展包, VSCode 可以提供代码补全、调试、测试等丰富的开发功能, 适合 Java 项目的开发。
- Java (JDK): Java Development Kit (JDK) 是 Java 开发的核心工具包, 包含了 Java 编译器、Java 运行时环境 (JRE) 以及一系列开发工具。JDK 提供了 Java 程序开发所需的所有基础组件, 确保代码能够在不同平台上运行。
- Maven: Maven 是一个强大的项目管理和构建工具, 主要用于 Java 项目。它通过 POM (Project Object Model) 文件来管理项目的依赖、构建配置和生命周期。Maven 能够自动下载所需的依赖库, 并提供了标准的项目结构, 简化了项目的构建和部署过程。

3.3 前端

主要使用 Vue3 作为框架, Node.js 作为开发环境, 同时使用 Element-plus 的 UI 组件。

- Vue3: Vue3 是一个流行的 JavaScript 框架, 用于构建用户界面和单页应用。Vue3 提供了响应式数据绑定、组件化开发等特性, 使得前端开发更加高效和灵活。通过组合式 API, 开发者可以更好地组织和复用代码。

- Node.js: Node.js 是一个基于 Chrome V8 引擎的 JavaScript 运行时环境, 允许开发者使用 JavaScript 编写服务器端代码。Node.js 提供了丰富的模块和工具, 支持前端开发中的包管理和构建工具, 如 npm 和 Webpack。
- Element-plus: Element-plus 是一个基于 Vue3 的 UI 组件库, 提供了丰富的 UI 组件和样式, 帮助开发者快速构建美观且功能丰富的前端界面。Element-plus 的组件易于使用和定制, 适合用于管理系统的开发。

3.4 后端

- Java: Java 具有跨平台、面向对象、强类型等特性, 在企业级应用开发中非常流行, 尤其适合构建后端系统。Java 的稳定性和丰富的生态系统使其成为后端开发的首选语言之一。
- JDBC: Java Database Connectivity (JDBC) 是 Java 中用于连接和操作数据库的标准 API。通过 JDBC, Java 程序可以与各种关系型数据库 (如 MySQL) 进行交互, 执行 SQL 语句并处理结果集。JDBC 提供了数据库操作的统一接口, 简化了数据库访问的复杂性。
- REST-API: REST (Representational State Transfer) 是一种基于 HTTP 协议的架构风格, 用于设计网络应用程序的 API。REST-API 通过 HTTP 方法 (如 GET、POST、PUT、DELETE) 来操作资源, 通常返回 JSON 或 XML 格式的数据。REST-API 具有简单、灵活、易于扩展的特点, 适合用于前后端分离的系统中。

3.5 数据库

- MySQL: 广泛使用的关系型数据库管理系统, 支持多用户、多线程操作, 具有高性能、高可靠性的特点。MySQL 提供了丰富的 SQL 功能, 支持事务处理、存储过程、触发器等高级特性, 适合用于存储和管理图书管理系统的数据。

4 模块设计

该部分主要分为前端、后端与前后端 API 的制定。

4.1 前端

- 在 `/front-end/src/router/index.js` 文件中, 定义了前端应用的路由配置。Vue Router 是 Vue.js 官方的路由管理工具, 用于实现单页面应用 (SPA) 中的页面跳转和导航。通过 `createRouter` 和 `createWebHistory` 方法创建了一个路由实例, 其中 `createWebHistory` 使用了 HTML5 的 History API 来实现无刷新页面跳转。

路由配置中定义了三个主要路径: `/book`、`/card` 和 `/borrow`, 分别对应图书管理、借书证管理和借书记录查询三个功能模块。默认情况下, 访问根路径 `/` 时会重定向到 `/book`, 即图书管理页面。每个路径都映射到一个 Vue 组件, 分别是 `Book.vue`、`Card.vue` 和 `Borrow.vue`, 这些组件负责渲染对应的页面内容并处理相关逻辑。

- 在 `/front-end/src/App.vue` 文件中, 定义了整个应用的入口页面布局。页面采用了 Element Plus 提供的容器组件 (`el-container`、`el-header`、`el-aside` 和 `el-main`) 来构建整体

结构。页面左侧是一个侧边栏 (el-aside)，使用 Element Plus 的菜单组件 (el-menu) 实现了导航功能。菜单中包含了三个选项：图书管理、借书证管理和借书记录查询，每个选项都绑定了一个路由路径，点击后会跳转到对应的页面。侧边栏的样式和交互行为通过 el-menu 的属性和事件进行配置，例如高亮当前选中的菜单项、设置菜单的背景颜色和文字颜色等。页面的主内容区 (el-main) 通过 <RouterView> 组件动态渲染当前路由对应的页面内容。为了支持长内容的滚动，主内容区使用了 Element Plus 的滚动条组件 (el-scrollbar)，确保页面内容在超出可视区域时能够平滑滚动。

具体的功能实现位于 /front-end/src/components 目录下的三个 Vue 组件文件中：Book.vue、Borrow.vue 和 Card.vue。每个组件对应一个功能模块，分别负责处理图书管理、借书记录查询和借书证管理的相关逻辑。这些组件通过 Vue 的组合式 API 编写，利用 ref、reactive、computed 等特性管理组件的状态和逻辑。此外，组件中还会通过 Axios 或其他 HTTP 客户端与后端 REST API 进行交互，获取数据并更新页面内容。

4.1.1 Book.vue

Book.vue 是图书管理系统的核心组件之一，负责实现图书管理的各项功能。该组件通过 Vue 3 的组合式 API 编写，结合 Element Plus 的 UI 组件库，提供了丰富的用户交互界面和功能，示例如下：



1. 页面布局与样式

- 页面主体部分采用弹性布局 (flex)，将功能卡片以网格形式排列。每个功能卡片都是一个 el-button 组件，点击后会触发相应的对话框或操作。
- 功能卡片的样式通过 box-shadow 实现了立体效果，并使用了 Element Plus 的图标组件 (如 Notebook、DocumentChecked 等) 来增强视觉效果。

2. 功能卡片

- **借书功能**: 点击“我要借书”卡片后, 会弹出一个对话框, 用户需要输入书号、卡号和借书时间。借书时间通过 `el-date-picker` 组件选择, 支持日期和时间的选择。点击“确定”按钮后, 会通过 `Axios` 向后端发送借书请求。
- **还书功能**: 点击“我要还书”卡片后, 会弹出一个对话框, 用户需要输入书号、卡号和还书时间。还书时间同样通过 `el-date-picker` 组件选择。点击“确定”按钮后, 会通过 `Axios` 向后端发送还书请求。
- **查询图书**: 点击“查询图书”卡片后, 会弹出一个对话框, 用户可以根据类别、书名、出版社、出版年份、作者、价格等条件进行查询。查询结果会以表格形式展示在页面下方, 支持按不同字段排序。
- **图书入库**: 点击“图书入库”卡片后, 会弹出一个对话框, 用户可以输入图书的类别、书名、出版社、出版年份、作者、价格和库存信息。点击“确定”按钮后, 会通过 `Axios` 向后端发送入库请求。
- **图书批量入库**: 点击“图书批量入库”卡片后, 会弹出一个对话框, 用户可以通过表格形式批量录入多本图书的信息。每行代表一本图书, 支持动态增加和删除行。点击“确定”按钮后, 会通过 `Axios` 向后端发送批量入库请求。
- **删除图书**: 点击“删除图书”卡片后, 会弹出一个对话框, 用户需要输入要删除的图书的书号。点击“确定”按钮后, 会通过 `Axios` 向后端发送删除请求。
- **修改库存**: 点击“图书库存修改”卡片后, 会弹出一个对话框, 用户需要输入书号和库存变化量。点击“确定”按钮后, 会通过 `Axios` 向后端发送库存修改请求。
- **修改图书信息**: 点击“图书信息修改”卡片后, 会弹出一个对话框, 用户可以修改图书的类别、书名、出版社、出版年份、作者和价格等信息。点击“确定”按钮后, 会通过 `Axios` 向后端发送信息修改请求。

3. 对话框与表单

- 每个功能卡片点击后都会弹出一个对话框 (`el-dialog`), 对话框中包含相应的表单 (`el-input`、`el-date-picker` 等), 用户需要填写必要的信息。
- 表单的输入项通过 `v-model` 与 `Vue` 的数据属性进行双向绑定, 确保用户输入的数据能够实时更新到组件的状态中。
- 对话框的“确定”按钮会根据表单的填写情况动态禁用或启用, 确保用户必须填写所有必填项后才能提交请求。

4. 数据交互

- 所有与后端的数据交互都通过 `Axios` 实现。`Axios` 是一个基于 `Promise` 的 `HTTP` 客户端, 支持发送 `GET`、`POST` 等请求, 并处理响应数据。
- 请求的 `URL` 根据功能不同而变化, 例如借书功能的 `URL` 为 `/book?action=borrow`, 还书功能的 `URL` 为 `/book?action=return`, 查询功能的 `URL` 为 `/book?type=records` 等。
- 请求的参数通过 `JavaScript` 对象的形式传递给后端, 后端返回的响应数据会通过 `ElMessage` 组件显示给用户, 例如成功提示或错误信息。

5. 查询结果展示

- 查询图书功能的结果会以表格形式展示在页面下方。表格使用 `el-table` 组件实现，支持按不同字段排序（如类别、书名、出版社等）。
- 表格的高度固定为 500px，超出部分可以通过滚动条查看。表格的列宽根据内容自动调整，确保数据能够完整显示。

6. 状态管理

- 组件的状态通过 Vue 的 `data` 函数进行管理，包括对话框的可见性、表单的输入值、查询结果等。
- 每个功能的状态都独立管理，例如借书功能的 `newBorrowVisible` 控制借书对话框的显示与隐藏，`newBorrowInfo` 存储用户输入的借书信息。

7. 工具方法

- `formatDateToString` 方法用于将日期对象转换为后端所需的字符串格式（如 YYYYMMDDHHmmss），确保日期数据能够正确传递。
- `deleteRow` 方法用于删除批量入库表格中的某一行数据。
- `onAddItem` 方法用于在批量入库表格中新增一行数据。

8. 样式与交互

- 组件的样式通过 `<style scoped>` 定义，确保样式只作用于当前组件，避免与其他组件冲突。
- 功能卡片的交互通过 `@click` 事件绑定，点击后会触发相应的对话框或操作。
- 对话框的关闭操作通过 `@click` 事件绑定到“取消”按钮，点击后会隐藏对话框并清空表单数据。

9. 错误处理

- 所有 `Axios` 请求都通过 `.then` 和 `.catch` 处理响应和错误。如果后端返回错误信息，会通过 `ElMessage.error` 显示给用户；如果操作成功，会通过 `ElMessage.success` 显示成功提示。

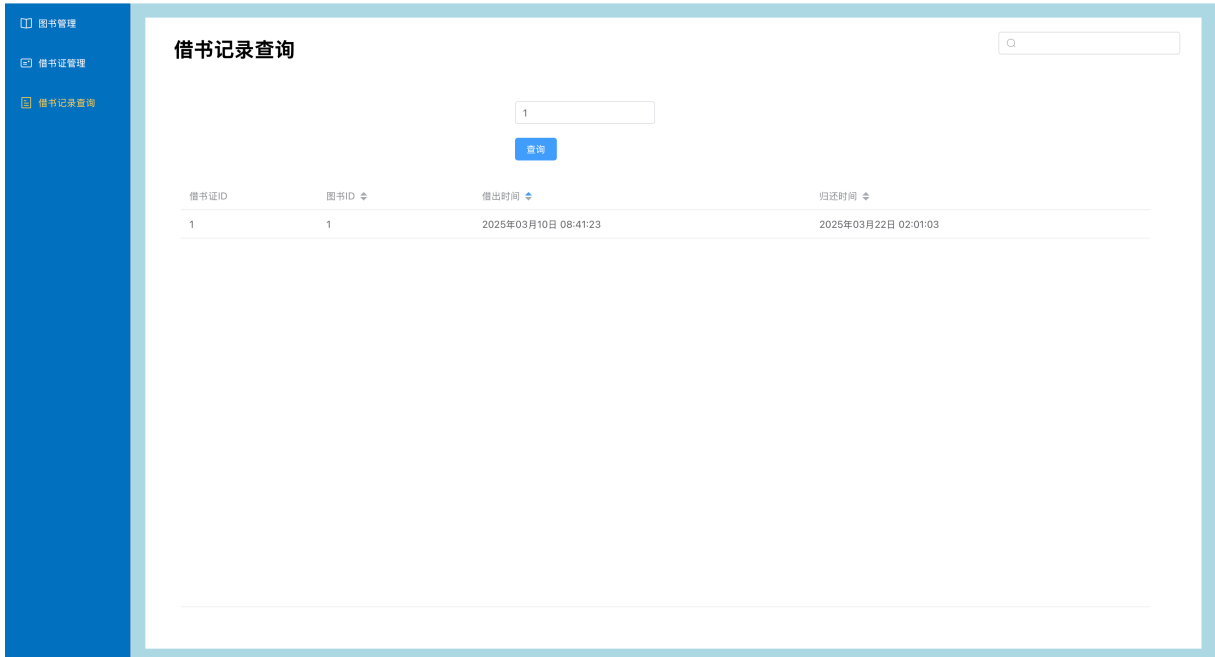
10. 代码结构

- 组件的代码结构分为模板部分（`<template>`）、脚本部分（`<script>`）和样式部分（`<style>`）。
- 模板部分定义了页面的布局和交互元素，脚本部分实现了业务逻辑和数据交互，样式部分定义了页面的视觉效果。

通过以上设计，`Book.vue` 实现了一个交互友好的图书管理界面，能够满足用户对图书管理的各项需求。

4.1.2 Borrow.vue

与 `Book.vue` 结构相似，`Borrow.vue` 是图书管理系统中用于查询借书记录的组件，提供了简洁且功能完善的借书记录查询界面。以下是 `Borrow.vue` 的主要功能模块及其实现细节：



1. 页面布局与样式

- 标题右侧提供了一个搜索框（`el-input`），用户可以在查询结果中进一步筛选借书记录。搜索框支持按图书 ID、借出时间或归还时间进行筛选。
- 页面主体部分包含一个查询框和一个结果表格。查询框用于输入借书证 ID，点击“查询”按钮后会显示对应的借书记录。

2. 查询功能

- 用户在查询框中输入借书证 ID 后，点击“查询”按钮会触发 `QueryBorrows` 方法。
- `QueryBorrows` 方法通过 `Axios` 向 `/borrow?type=records&cardId=<借书证 ID>` 发送 GET 请求，获取该借书证的所有借书记录。
- 如果后端返回错误信息，会通过 `ElMessage.error` 显示给用户；如果查询成功，借书记录会以表格形式展示在页面下方。

3. 结果表格

- 查询结果通过 `el-table` 组件展示，表格包含四列：借书证 ID、图书 ID、借出时间和归还时间。
- 表格支持按图书 ID、借出时间和归还时间进行排序，默认按借出时间升序排列。
- 表格的高度固定为 600px，超出部分可以通过滚动条查看。表格的列宽根据内容自动调整，确保数据能够完整显示。

4. 时间格式化

- 后端返回的借出时间和归还时间是数字格式（如 20250304092613），通过 `reflect` 方法将其转换为更易读的格式（如 2025 年 03 月 04 日 09:26:13）。
- 如果归还时间为 0，表示图书尚未归还，`reflect` 方法会返回“未归还”。

5. 状态管理

- 组件的状态通过 Vue 的 data 函数进行管理，包括查询结果的展示状态 (isShow)、表格数据 (tableData)、借书证 ID (cardID) 和搜索内容 (toSearch)。
- 查询结果的展示状态通过 isShow 控制，初始状态下表格不显示，查询成功后设置为 true。

6. 交互细节

- 查询框的“查询”按钮通过 @click 事件绑定到 QueryBorrows 方法，点击后会触发查询操作。
- 搜索框的输入内容通过 v-model 与 toSearch 属性进行双向绑定，确保用户输入的内容能够实时更新到组件的状态中。

4.1.3 Card.vue

Card.vue 是图书管理系统中用于管理借书证的组件，界面示例如下：



1. 页面布局与样式

- 标题右侧提供了一个搜索框 (el-input)，用户可以根据借书证号、姓名或部门进行筛选。
- 页面主体部分采用弹性布局 (flex)，将借书证卡片以网格形式排列。每个卡片展示了借书证的基本信息，并提供了修改和删除操作按钮。

2. 借书证卡片

- 每个借书证卡片展示了以下信息：
 - **借书证号**：card.cardId
 - **姓名**：card.name
 - **部门**：card.department
 - **类型**：通过 reflection 方法将类型代码 (T 或 S) 转换为可读的“教师”或“学生”。
- 每个卡片底部提供了两个操作按钮：

- **修改**：点击后弹出修改对话框，允许用户修改借书证的姓名、部门和类型。
- **删除**：点击后弹出删除确认对话框，用户确认后删除该借书证。

3. 新建借书证

- 页面右下角提供了一个“新建借书证”按钮，点击后会弹出一个对话框。
- 对话框中包含三个输入项：
 - **姓名**：必填项，用户输入借书证持有人的姓名。
 - **部门**：必填项，用户输入借书证持有人所属的部门。
 - **类型**：下拉选择框，用户可以选择“教师”或“学生”。
- 点击“确定”按钮后，会通过 Axios 向后端发送新建借书证的请求。

4. 修改借书证

- 点击某个借书证卡片的“修改”按钮后，会弹出一个修改对话框。
- 对话框中预填充了当前借书证的信息，用户可以修改姓名、部门和类型。
- 点击“确定”按钮后，会通过 Axios 向后端发送修改借书证的请求。

5. 删除借书证

- 点击某个借书证卡片的“删除”按钮后，会弹出一个删除确认对话框。
- 用户确认后，会通过 Axios 向后端发送删除借书证的请求。

6. 数据交互

- 所有与后端的数据交互都通过 Axios 实现。Axios 是一个基于 Promise 的 HTTP 客户端，支持发送 GET、POST 等请求，并处理响应数据。
- 请求的 URL 根据功能不同而变化，例如：
 - 查询借书证：/card?type=records
 - 新建借书证：/card?action=register
 - 修改借书证：/card?action=modify
 - 删除借书证：/card?action=remove
- 请求的参数通过 JavaScript 对象的形式传递给后端，后端返回的响应数据会通过 EIMessage 组件显示给用户，例如成功提示或错误信息。

7. 状态管理

- 组件的状态通过 Vue 的 data 函数进行管理，包括借书证列表 (cards)、搜索内容 (toSearch)、新建借书证信息 (newCardInfo)、待修改借书证信息 (toModifyInfo) 等。
- 每个功能的状态都独立管理，例如新建借书证对话框的可见性通过 newCardVisible 控制，修改借书证对话框的可见性通过 modifyCardVisible 控制。

8. 工具方法

- reflection 方法用于将借书证类型代码 (T 或 S) 转换为可读的“教师”或“学生”。
- QueryCards 方法用于查询所有借书证，并将结果存储在 cards 列表中。
- ConfirmNewCard、ConfirmModifyCard 和 ConfirmRemoveCard 方法分别用于处理新建、修改和删除借书证的逻辑。

4.2 后端

后端主要分为 main 和 test 两部分，分别对应后端实现代码与测试代码。main 文件夹又可分为 java 与 resources 两部分，分别对应后端的实现与数据库的连接相关资源。

在 src/main/java/entities 中定义了 Book、Borrow、Card 三个数据对象，jsonHandlers 中定义了一些类，用于处理前端传来的 JSON 字符串并利用 Gson 进行解析。queries 中定义了 ApiResult 以及一些作为结果返回的数据对象，而 utils 中则存放了一些功能类，例如连接数据库的 DatabaseConnector。

后端的核 心部分是 LibraryManagementSystem 接口，具体的实现代码位于 LibraryManagementSystemImpl 类中，即后端接受前端数据并与数据库交互、返回结果的部分。

功能模块详解如下：

4.2.1 添加单本图书 (storeBook(Book book))

该方法用于向数据库中添加一本新书。其主要逻辑如下：

- 首先，通过 SQL 查询检查数据库中是否已存在相同属性（分类、标题、出版社、出版年份、作者）的图书。如果存在，则返回失败结果，提示“书库中已存在此书”。
- 如果不存在，则构造一个 SQL 插入语句，将新书的详细信息（分类、标题、出版社、出版年份、作者、价格、库存）插入到数据库的 book 表中。
- 插入成功后，通过 getGeneratedKeys 方法获取数据库自动生成的图书 ID，并将其设置到 Book 对象的 bookId 属性中。
- 最后提交事务，返回成功结果，提示“Successfully stored book!”。

4.2.2 批量添加图书 (storeBook(List<Book> books))

该方法用于批量添加多本图书。其主要逻辑如下：

- 遍历待添加的图书列表，对每本书都执行与单本图书相同的重复性检查。如果发现任何一本重复，则返回失败结果，提示“书库中已存在此书”。
- 如果所有图书都通过检查，则使用 addBatch 和 executeBatch 方法将所有图书批量插入到数据库中。
- 插入成功后，通过 getGeneratedKeys 方法获取数据库自动生成的图书 ID，并依次设置到每本图书对象的 bookId 属性中。
- 最后提交事务，返回成功结果，提示“Successfully stored books!”。

4.2.3 增加图书库存 (incBookStock(int bookId, int deltaStock))

该方法用于增加指定图书的库存数量。其主要逻辑如下：

- 首先检查指定的图书 ID 是否存在。如果不存在，则返回失败结果，提示“图书不存在”。
- 然后检查增加库存后是否会使得库存数量变为负值。如果是，则返回失败结果，提示“图书库存不能为负值”。
- 如果检查通过，则更新数据库中该图书的库存数量，通过 SQL 语句 UPDATE book SET stock = stock + ? WHERE book_id = ? 来实现。
- 最后提交事务，返回成功结果，提示“Successfully updated book stock!”。

4.2.4 删除图书 (**removeBook(int bookId)**)

该方法用于从数据库中删除一本指定的图书。其主要逻辑如下：

- 首先检查指定的图书 ID 是否存在。如果不存在，则返回失败结果，提示“不存在此图书”。
- 然后检查是否有读者尚未归还该图书。如果有，则返回失败结果，提示“图书尚未归还”，不允许删除。
- 如果检查通过，则从数据库中删除该图书记录，通过 SQL 语句 `DELETE FROM book WHERE book_id = ?` 来实现。
- 最后提交事务，返回成功结果，提示“Successfully removed book!”。

4.2.5 修改图书信息 (**modifyBookInfo(Book book)**)

该方法用于修改指定图书的信息。其主要逻辑如下：

- 首先检查指定的图书 ID 是否存在。如果不存在，则返回失败结果，提示“图书不存在”。
- 然后构造一个 SQL 更新语句，将更新后的图书信息（分类、标题、出版社、出版年份、作者、价格）写入到数据库中，覆盖原有记录。
- 更新成功后，提交事务，返回成功结果，提示“Successfully modified book info!”。

4.2.6 借书 (**borrowBook(Borrow borrow)**)

该方法用于处理读者借书的请求。其主要逻辑如下：

- 首先检查指定的图书 ID 和借书证 ID 是否存在。如果任意一个不存在，则返回失败结果，提示“不存在此书”或“不存在此借书证”。
- 然后检查该图书的库存是否充足。如果不充足，则返回失败结果，提示“图书库存不足”。
- 接着检查读者是否已经借阅了该书且尚未归还。如果是，则返回失败结果，提示“您已借阅过此书但未归还”。
- 如果检查通过，则将借书记录插入到数据库的 borrow 表中，并将该书的库存数量减一，通过 SQL 语句 `UPDATE book SET stock = stock - 1 WHERE book_id = ?` 来实现。
- 最后提交事务，返回成功结果，提示“借书成功”。

4.2.7 还书 (**returnBook(Borrow borrow)**)

该方法用于处理读者还书的请求。其主要逻辑如下：

- 首先检查指定的图书 ID 和借书证 ID 是否存在。如果任意一个不存在，则返回失败结果，提示“不存在此书”或“不存在此借书证”。
- 然后检查读者是否借阅了该书且尚未归还。如果没有，则返回失败结果，提示“您未借阅过此书”。
- 接着检查还书时间是否晚于借书时间。如果不是，则返回失败结果，提示“还书时间必须晚于借书时间”。
- 如果检查通过，则将借书记录中的还书时间更新为当前时间，并将该书的库存数量加一，通过 SQL 语句 `UPDATE book SET stock = stock + 1 WHERE book_id = ?` 来实现。
- 最后提交事务，返回成功结果，提示“成功还书”。

4.2.8 查询借阅历史 (showBorrowHistory(int cardId))

该方法用于查询指定借书证的借阅历史。其主要逻辑如下：

- 首先检查指定的借书证 ID 是否存在。如果不存在，则返回失败结果，提示“借书证不存在”。
- 然后从数据库中查询该借书证的所有借阅记录，包括图书信息、借书时间和还书时间等，通过 SQL 语句 `SELECT card_id, book_id, category, title, press, publish_year, author, price, borrow_time, return_time FROM card NATURAL JOIN borrow NATURAL JOIN book WHERE card_id = ? ORDER BY borrow_time DESC, book_id ASC` 来实现。
- 将查询结果封装为 `BorrowHistories` 对象并返回。

4.2.9 注册借书证 (registerCard(Card card))

该方法用于注册一张新的借书证。其主要逻辑如下：

- 首先检查数据库中是否已存在相同属性（姓名、部门、类型）的借书证。如果存在，则返回失败结果，提示“借书证已经存在”。
- 如果不存在，则将新的借书证信息插入到数据库的 `card` 表中，并获取数据库自动生成的借书证 ID，将其设置到 `Card` 对象的 `cardId` 属性中。
- 最后提交事务，返回成功结果，提示“Successfully registered card!”。

4.2.10 删除借书证 (removeCard(int cardId))

该方法用于从数据库中删除一张指定的借书证。其主要逻辑如下：

- 首先检查指定的借书证 ID 是否存在。如果不存在，则返回失败结果，提示“借书证不存在”。
- 然后检查该借书证是否有尚未归还的图书。如果有，则返回失败结果，提示“当前借书证有未归还图书”，不允许删除。
- 如果检查通过，则从数据库中删除该借书证记录，通过 SQL 语句 `DELETE FROM card WHERE card_id = ?` 来实现。
- 最后提交事务，返回成功结果，提示“Successfully removed card!”。

4.2.11 修改借书证信息 (modifyCardInfo(Card card))

该方法用于修改指定借书证的信息。其主要逻辑如下：

- 首先检查指定的借书证 ID 是否存在。如果不存在，则返回失败结果，提示“借书证不存在”。
- 然后将更新后的借书证信息写入到数据库中，覆盖原有记录，通过 SQL 语句 `UPDATE card SET name = ?, department = ?, type = ? WHERE card_id = ?` 来实现。
- 更新成功后，提交事务，返回成功结果，提示“Successfully modified card info!”。

4.2.12 查询所有借书证 (showCards())

该方法用于查询数据库中所有的借书证信息。其主要逻辑如下：

- 从数据库中查询 `card` 表中的所有记录，通过 SQL 语句 `SELECT * FROM card ORDER BY card_id ASC` 来实现。
- 将查询结果封装为 `CardList` 对象并返回。

以上所有的方法均通过 JDBC 与数据库进行交互, 具体的逻辑为, 先通过 PreparedStatement 类构建一个对象, 用? 表示占位符, 即待插入的参数。之后从方法参数中获得查询语句需要的参数, 并通过 setInt 或其他方法组成一个完整的 sql 语句。最后调用 executeQuery 等方法执行 sql 语句并使用一个 ResultSet 对象接收查询结果, 对结果再进行相应处理。

4.3 前后端 API

前后端之间使用 REST 风格的 API 传递信息, 主要为 json 字符串。对于前端而言, 如果是查询类型的事件, 使用 GET 请求; 如果是执行类型的事件, 例如更新数据库中的某一本书, 使用 POST 请求; OPTION 请求用于跨域预检。后端接收来自前端的请求, 使用 Gson 解析, 根据参数不同进行相应处理。

4.3.1 后端 API 设计

后端采用 RESTful 风格, 使用 GET 处理查询请求, POST 处理执行请求, 并提供 OPTIONS 以支持跨域预检。后端解析来自前端的请求, 分配路由 (具体的请求用哪个 handler 去处理), 然后查询数据库、返回数据等。

4.3.1.1 服务器初始化

后端服务器的启动由 Main.java 负责, 该类主要执行以下操作:

1. 解析数据库连接配置:

- 通过 ConnectConfig 读取数据库配置信息。
- 记录配置信息到日志中, 确保正确解析。

2. 初始化数据库连接:

- 通过 DatabaseConnector 建立数据库连接。
- 连接失败时, 记录错误日志并终止程序。

3. 创建业务逻辑层实例:

- LibraryManagementSystemImpl 作为核心逻辑层, 与数据库交互。

4. 启动 HTTP 服务器:

- 监听 3306 端口, 等待前端请求。
- 注册 BookHandler、BorrowHandler 和 CardHandler 以处理相关请求。

5. 注册关闭钩子:

- 确保服务器关闭时正确释放数据库连接, 避免资源泄漏。

4.3.1.2 BookHandler API

BookHandler.java 负责处理 /book 路径的 HTTP 请求。

4.3.1.2.1 GET 请求: 查询书籍

- 解析 URL 查询参数, 构造 BookQueryConditions。
- 调用 LibraryManagementSystem.queryBook 获取查询结果。
- 结果转换为 JSON 格式返回。

示例请求:

GET /book?type=records&title=Java

示例返回:

```
{
  "records": [
    {"bookId": 1, "title": "Java Programming", "author": "John Doe", ...},
    {"bookId": 2, "title": "Advanced Java", "author": "Jane Smith", ...}
  ]
}
```

4.3.1.2.2 POST 请求: 执行操作

根据 action 参数执行不同操作:

- store: 添加单本书籍。
- incstock: 增加库存。
- storemulti: 批量添加书籍。
- remove: 删除书籍。
- modify: 修改书籍信息。
- borrow: 借阅书籍。
- return: 归还书籍。

示例请求 (新增书籍):

```
POST /book
Content-Type: application/json
{
  "action": "store",
  "book": {"title": "New Book", "author": "Alice", "price": 39.99}
}
```

示例返回:

```
{"success": "图书入库成功"}
```

4.3.1.3 BorrowHandler API

BorrowHandler.java 负责 /borrow 路径的 HTTP 请求。

4.3.1.3.1 GET 请求: 查询借阅记录

- 解析 cardId, 查询该借书证的借阅历史。
- 通过 LibraryManagementSystem.showBorrowHistory 获取数据。
- 结果转换为 JSON 返回。

示例请求:

GET /borrow?type=records&cardId=123456

示例返回:

```
{
  "records": [
    {"bookId": 1, "title": "Java Programming", "borrowTime": 1678901234, "returnTime": 0}
  ]
}
```

```
]
}
```

4.3.1.4 CardHandler API

CardHandler.java 负责 /card 路径的 HTTP 请求。

4.3.1.4.1 GET 请求：查询借书证信息

- 查询所有借书证并返回 JSON 结构数据。

4.3.1.4.2 POST 请求：执行操作

根据 action 参数执行不同操作：

- register：注册新借书证。
- remove：删除借书证。
- modify：修改借书证信息。

示例请求（注册借书证）：

```
POST /card
Content-Type: application/json
{
  "action": "register",
  "card": {"name": "Alice", "department": "Computer Science", "type": "S"}
}
```

示例返回：

```
{"success": "注册借书证成功"}
```

4.3.1.5 跨域请求处理

所有 Handler 类都支持 OPTIONS 请求，以实现跨域请求的预检。

4.3.2 前端 API 设计

前端采用 Vue.js 框架，使用 axios 进行 HTTP 请求，并结合 Element Plus 进行 UI 交互。

4.3.2.1 Book.vue API

前端 Book.vue 组件用于管理书籍相关的操作，包括查询、借阅、归还、入库、删除、修改库存等，以下选取一些操作进行讲解。

4.3.2.1.1 借阅书籍

- 方法: ConfirmNewBorrow
- 请求类型: POST
- 请求路径: /book
- 请求参数:
 - action: "borrow"
 - borrow: { bookId, cardId, borrowTime, returnTime }
- 响应处理:
 - 如果返回 error，则弹出错误消息。
 - 如果成功，则关闭对话框并清空输入框。

示例请求:

```
{
  "action": "borrow",
  "borrow": {
    "bookId": "1",
    "cardId": "1001",
    "borrowTime": "20250324123045",
    "returnTime": "0"
  }
}
```

4.3.2.1.2 归还书籍

- 方法: ConfirmNewReturn
- 请求类型: POST
- 请求路径: /book
- 请求参数:
 - action: "return"
 - borrow: { bookId, cardId, borrowTime, returnTime }

4.3.2.1.3 查询书籍

- 方法: ConfirmQuery
- 请求类型: GET
- 请求路径: /book?type=records&[参数]
- 可选参数:
 - category, title, press, min-publish-year, max-publish-year, author, minprice, maxprice, sortby, sortorder

示例请求:

GET /book?type=records&title=Java&sortby=price&sortorder=asc

示例返回:

```
{
  "records": [
    { "bookId": 1, "title": "Java Programming", "author": "John Doe", ... }
  ]
}
```

4.3.2.1.4 入库书籍

- 方法: ConfirmStore
- 请求类型: POST
- 请求路径: /book
- 请求参数:
 - action: "store"
 - book: { bookId, category, title, press, publishYear, author, price, stock }

4.3.2.1.5 批量入库

- 方法: ConfirmBatchStore
- 请求类型: POST
- 请求路径: /book
- 请求参数:
 - action: "storemulti"
 - books: [{ book1 }, { book2 }]

4.3.2.2 Borrow.vue API

Borrow.vue 组件负责查询借阅记录。

4.3.2.2.1 查询借阅记录

- 方法: QueryBorrows
- 请求类型: GET
- 请求路径: /borrow?type=records&cardId=[cardId]
- 响应处理:
 - 将返回的 borrowTime 和 returnTime 格式化。
 - 填充借阅历史列表。

4.3.2.3 Card.vue API

Card.vue 组件用于管理借书证的增删改查。

4.3.2.3.1 新增借书证

- 方法: ConfirmNewCard
- 请求类型: POST
- 请求路径: /card
- 请求参数:
 - action: "register"
 - card: { name, department, type }

4.3.2.3.2 删除借书证

- 方法: ConfirmRemoveCard
- 请求类型: POST
- 请求路径: /card
- 请求参数:
 - action: "remove"
 - cardID: [cardId]

4.3.2.3.3 修改借书证信息

- 方法: ConfirmModifyCard
- 请求类型: POST
- 请求路径: /card
- 请求参数:

- action: "modify"
- card: { id, name, department, type }

4.3.2.3.4 查询借书证

- 方法: QueryCards
- 请求类型: GET
- 请求路径: /card?type=records
- 响应处理:
 - 获取所有借书证信息并填充列表。

示例请求:

GET /card?type=records

示例返回:

```
{
  "records": [
    { "cardId": 1001, "name": "Alice", "department": "Computer Science", "type": "S" }
  ]
}
```

4.3.2.4 前端 API 交互机制

1. **数据存储**: 使用 data() 维护页面状态。
2. **事件绑定**: 通过 methods 触发 axios 请求。
3. **UI 交互**: 结合 Element Plus 进行反馈。
4. **生命周期钩子**: 在 mounted() 中初始化数据。

5 系统测试

对图书管理系统的测试分为正确性测试与功能性测试，其中正确性测试主要由后端的 java 代码实现，功能性测试则由前端的界面操作进行测试。

5.1 正确性测试

首先打开终端，输入 brew services start mysql 启动 MySQL 服务：

```
Last login: Tue Mar 25 08:04:04 on ttys000
~$ brew services start mysql
==> Successfully started `mysql` (label: homebrew.mxcl.mysql)
```

之后在后端代码中找到/src/test/java/LibraryTest.java 这个文件，使用其中的代码进行测试。测试代码由框架提供，主要对后端 LibraryManagementSystem 类中的各个方法进行综合测试，例如多次借书还书，观察数据库中数据是否符合预期。

下表详细介绍了各个测试的主要测试内容：

测试名称	测试目的	主要验证点
bookRegisterTest	测试图书入库功能	验证图书唯一性约束 批量入库的正确性 重复图书检测 查询结果准确性
incBookStockTest	测试图书库存增减	库存增减边界条件 非法库存值处理 并发库存修改
bulkRegisterBookTest	测试批量图书入库	批量入库原子性 重复记录处理 事务回滚机制
removeBookTest	测试图书删除功能	被借阅图书删除限制 批量删除正确性 删除后查询一致性
modifyBookTest	测试图书信息修改	多字段更新验证 唯一性约束保持 库存字段保护
queryBookTest	测试图书查询功能	单条件/组合条件查询 模糊匹配 排序功能 分页结果
borrowAndReturnBookTest	测试借阅/归还流程	借阅状态验证 时间有效性检查 库存同步更新 借阅历史记录
parallelBorrowBookTest	测试并发借阅控制	并发冲突处理 库存一致性 线程安全

测试名称	测试目的	主要验证点
registerAndShowAndRemove-CardTest	测试读者卡管理	卡号唯一性 借阅关联检查 批量操作正确性

点击左侧的绿色三角形，运行全部测试。也可以下滑找到单个测试进行单独测试，这里使用全部测试：

```

17  import java.util.stream.Collectors;
18  import java.util.stream.Stream;
19
20  Click to run tests, right click for more options
21
22  private DatabaseConnector connector;
23  private LibraryManagementSystem library;
24
25  private static ConnectConfig connectConfig = null;

```

点击下方工具栏中的 TEST RESULT 可以查看测试结果，如图所示。如果有测试没有通过，会显示红色叉叉并相应报错。这里全部通过：

```

%TESTC 9 v2
%TSTTREE1,LibraryTest,true,9,false,-1,LibraryTest,,
%TSTTREE2,borrowAndReturnBookTest(LibraryTest),false,1,false,-1,borrowAndReturnBookTest(LibraryTest),,
%TSTTREE3,bulkRegisterBookTest(LibraryTest),false,1,false,-1,bulkRegisterBookTest(LibraryTest),,
%TSTTREE4,modifyBookTest(LibraryTest),false,1,false,-1,modifyBookTest(LibraryTest),,
%TSTTREE5,bookRegisterTest(LibraryTest),false,1,false,-1,bookRegisterTest(LibraryTest),,
%TSTTREE6,incBookStockTest(LibraryTest),false,1,false,-1,incBookStockTest(LibraryTest),,
%TSTTREE7,queryBookTest(LibraryTest),false,1,false,-1,queryBookTest(LibraryTest),,
%TSTTREE8,registerAndShowAndRemoveCardTest(LibraryTest),false,1,false,-1,registerAndShowAndRemoveCardTest(LibraryTest),,
%TSTTREE9,removeBookTest(LibraryTest),false,1,false,-1,removeBookTest(LibraryTest),,
%TSTTREE10,parallelBorrowBookTest(LibraryTest),false,1,false,-1,parallelBorrowBookTest(LibraryTest),,
%TESTS 2,borrowAndReturnBookTest(LibraryTest)
%TESTE 2,borrowAndReturnBookTest(LibraryTest)
%TESTS 3,bulkRegisterBookTest(LibraryTest)
%TESTE 3,bulkRegisterBookTest(LibraryTest)
%TESTS 4,modifyBookTest(LibraryTest)
%TESTE 4,modifyBookTest(LibraryTest)
%TESTS 5,bookRegisterTest(LibraryTest)
%TESTE 5,bookRegisterTest(LibraryTest)
%TESTS 6,incBookStockTest(LibraryTest)
%TESTE 6,incBookStockTest(LibraryTest)
%TESTS 7,queryBookTest(LibraryTest)
%TESTE 7,queryBookTest(LibraryTest)
%TESTS 8,registerAndShowAndRemoveCardTest(LibraryTest)
%TESTE 8,registerAndShowAndRemoveCardTest(LibraryTest)
%TESTS 9,removeBookTest(LibraryTest)
%TESTE 9,removeBookTest(LibraryTest)
%TESTS 10,parallelBorrowBookTest(LibraryTest)
%TESTE 10,parallelBorrowBookTest(LibraryTest)
%RUNTIMES251

```

5.2 功能性测试

功能性测试主要由前端界面进行测试。依次点击页面中的各个功能按钮，例如“我要借书”，输入相关借书信息，观察前后端与数据库是否按照预期运行。

与正确性测试相同，先使用 `brew services start mysql` 启动 MySQL 服务，之后点击后端 `src/main/java/Main.java` 中的 `run` 按钮启动后端服务，终端输出正在运行的信息，表明后端已正常运行并正常连接到数据库：

```

~/Desktop/Library-Management-System/ [main] /usr/bin/env /Library/Java/JavaVirtualMachines/jdk-23.jdk/Contents/Home/bin/java @/var/folders/x2/0q9d154d3j51ygvqw2yy39z00000gn/T/cp_73ztgcerxf8vknk2
y3he6i2n-argfile Main
Mar 25, 2025 3:34:25 PM Main main
INFO: Success to parse connect config. utils.ConnectConfig: {host='localhost', port='3306', user='root', password='XDSG94shuai@', db='library', type='DatabaseType{typeName='mysql', driverName='com.m
ysql.cj.jdbc.Driver'}}
Mar 25, 2025 3:34:25 PM Main main
INFO: Server started on port 3306.

```

之后将当前目录切换到前端目录，然后使用 `npm run dev` 命令启动前端服务，再点击相应的 localhost url，即可在浏览器中查看前端页面。

```

~ /Desktop/Library-Management-System/ [main] cd front-end
~ /Desktop/Library-Management-System/front-end/ [main] npm run dev

> librarymanagementsystem@0.0.0 dev
> vite

VITE v5.4.14 ready in 263 ms
→ Local:   http://localhost:5431/
→ Network: use --host to expose
→ press h + enter to show help

```

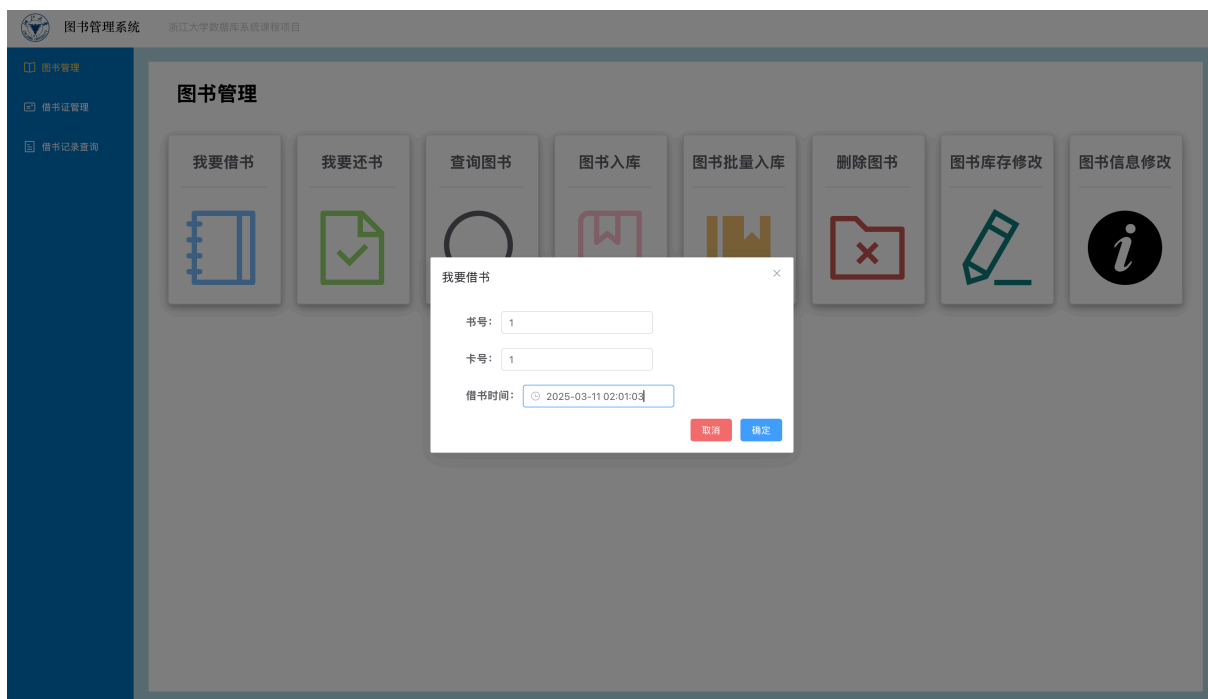
为保证测试顺利进行，在以下测试开始之前预先向数据库中插入了一些数据。

5.2.1 图书管理页面



5.2.1.1 我要借书

点击“我要借书”按钮，会弹出一个对话框。在对话框中输入借书的书号、借书证号以及借书时间：

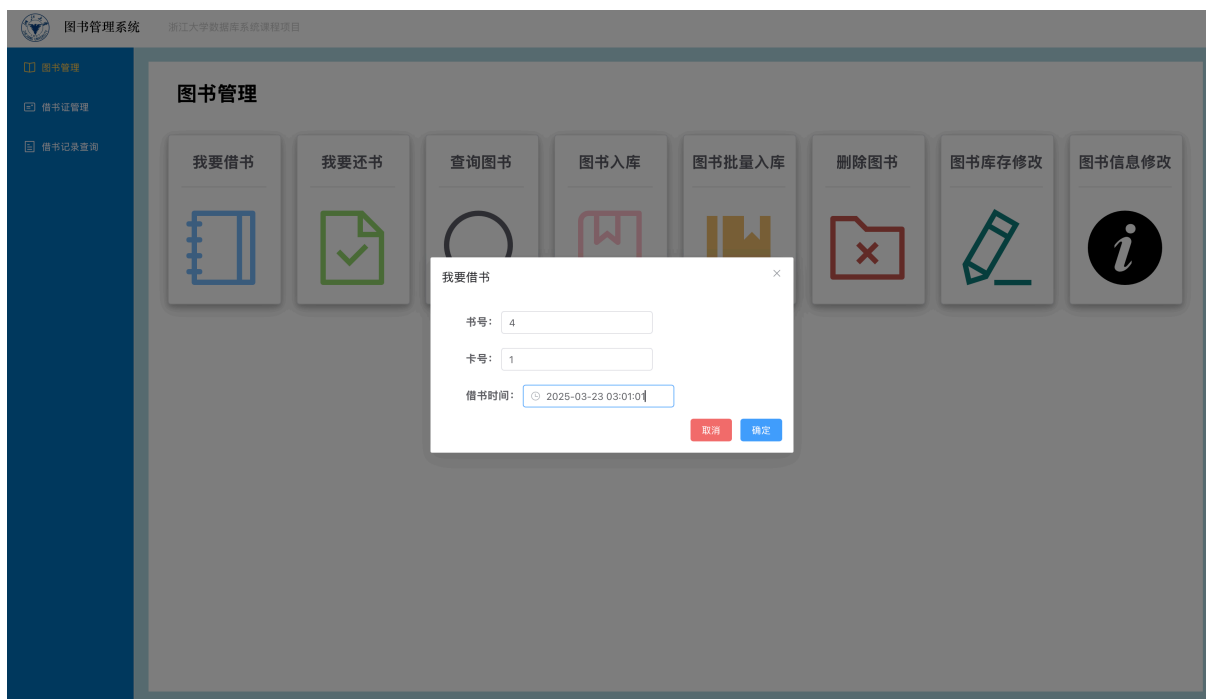


再点击“确定”按钮，即可完成借书操作（需要保证借书的书号、卡号都存在，不然会显示错误）。

若在未归还此书的情况下再次借书，会显示“借书失败，您已借过此书但未归还”，并阻止借书操作，如图所示：

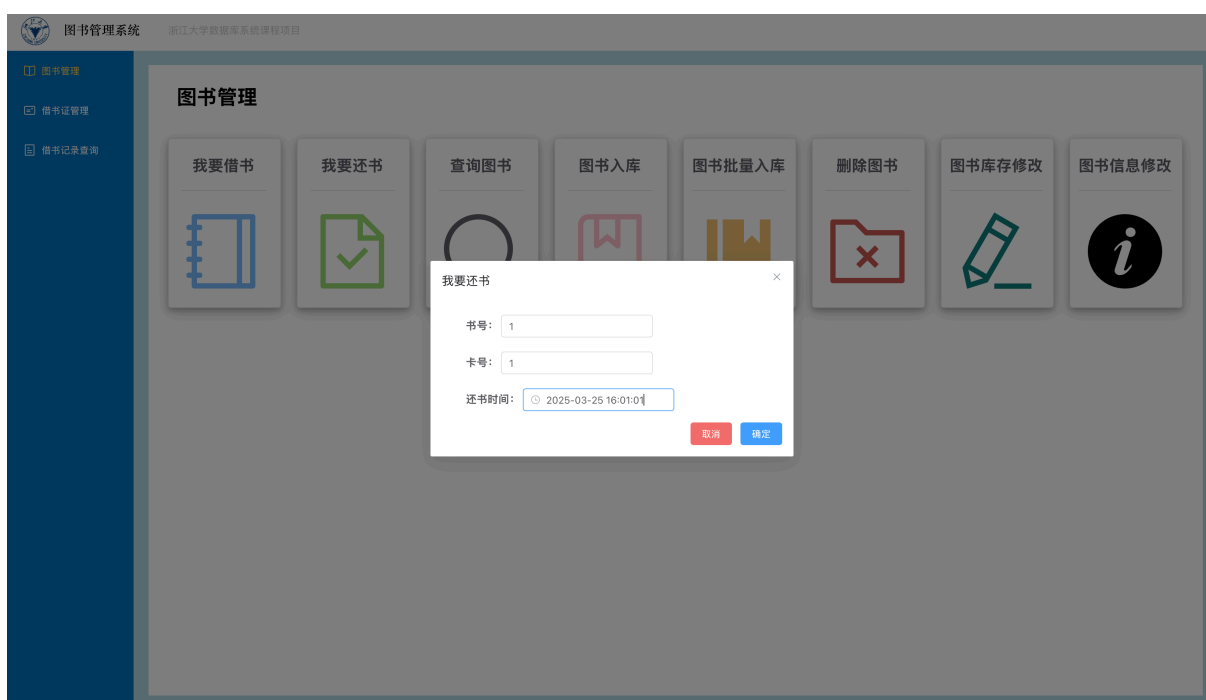


再借另外一本书，借书成功：



5.2.1.2 我要还书

点击“我要还书”按钮，并在弹出的对话框中填入书号、卡号以及还书时间，即可完成还书：



还书操作需要保证书号存在、卡号存在，且该书之前已经被借阅过未归还，否则会弹窗显示还书失败。

如果以上要求都满足，则还书成功：

✔ 还书成功

查询图书

图书入库

图书批量入库

5.2.1.3 查询图书

点击“查询图书”按钮，会弹出一个对话框。在框中填入查询条件，例如精确查询到类别名，并选择排序依据和排序方式（可选，默认按书号升序排列）。

该功能模块要求为从查询条件\<类别点查(精确查询)，书名点查(模糊查询)，出版社点查(模糊查询)，年份范围查，作者点查(模糊查询)，价格范围差>中随机选取 N 个条件，并随机选取一个排序列和顺序，在实际测试中选择类别为“计算机”，出版社为“机械工业”两个查询条件，并选择按出版年份降序排列进行测试：

查询图书 (至少输入一个查询条件)

类别: 计算机

书名: 模糊查询

出版社: 机械工业

最小年份: 选择年份

最大年份: 选择年份

作者: 模糊查询

最低价格:

最高价格:

排序依据: 出版年份

排序方式: 降序

取消 确定

点击“确定”进行查询，会出现提示“查询成功”的弹窗并在功能按钮下方列出查询结果，如图所示：

浙江大学数据库系统课程项目

图书管理

我要借书

我要还书

查询图书

图书入库

图书批量入库

删除图书

图书库存修改

图书信

书号	类别	书名	出版社	出版年份	作者	价格	库存
7	计算机	C程序设计基础与实验	机械工业	2014	John	59.99	90
6	计算机	数据结构基础	机械工业	2009	Tom	79.99	30
1	计算机	计算机系统原理	机械工业	1999	Michael	99.99	24
5	计算机	数据库系统原理	机械工业	1997	Mark	109.99	38

5.2.1.4 图书入库

点击“图书入库”，在弹出的对话框中填写图书的基本信息，如类别，书名等，即可实现单本图书入库：

我要还书

查询图书

图书入库

图书批量入库

删除图书

单本图书入库

类别：

书名：

出版社：

出版年份：

作者：

价格：

库存：

取消

确定

点击“确定”确认入库操作，若入库成功（书库中没有相同的书）会弹窗提示：

✓ 图书入库成功

5.2.1.5 图书批量入库

点击“图书批量入库”按钮，会显示一个初始为空的对话框。点击底部的“增加”可以新建一条插入数据，之后按照要求填写入库图书的信息：

类别	书名	出版社	出版年份	作者	价格	库存	操作
计算机	计算机网!	浙江大学	2018	不知道	89.99	14	删除
文学	中国文学!	文学院	2009	神秘教授	34.99	18	删除
哲学	苏菲的世!	哲学院	1998	Tina	20	33	删除

增加

取消 确定

全部填写完成之后点击“确定”即可完成批量入库。需要注意批量入库的图书为一个整体，如果其中一本图书入库失败，则操作会全部回滚，所有图书都不能入库。如果成功批量入库，同样会弹窗提醒：

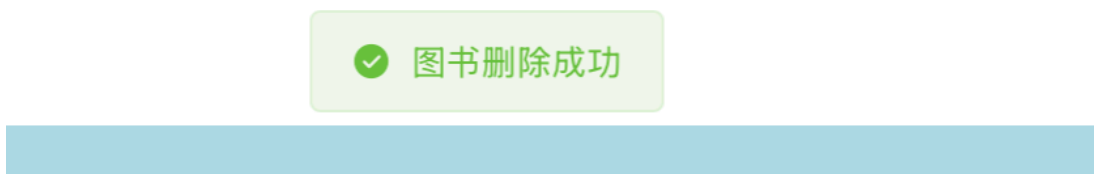
✓ 图书入库成功

5.2.1.6 删除图书

与之前操作相同，点击“删除图书”按钮，填写需要删除的图书的书号（这里删除书号为1的图书进行测试）：



删除成功后会同步更新数据库当中的数据，并在前端界面中弹窗提示：



5.2.1.7 图书库存修改

点击“图书库存修改”按钮，弹出一个对话框，输入需要修改的图书的书号以及变化量（可正可负，但修改后的库存不能为负值，否则会弹窗报错）：



进行图书查询以查看测试结果：

修改前：

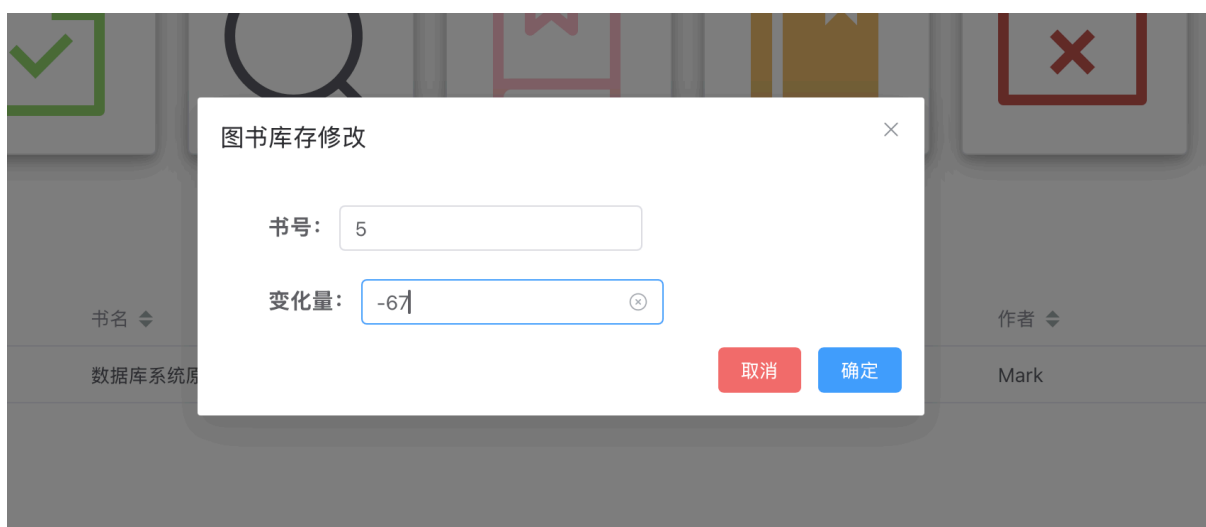
图书查询结果 隐藏							
书号	类别	书名	出版社	出版年份	作者	价格	库存
5	计算机	数据库系统原理	机械工业	1997	Mark	109.99	38

修改后：

图书查询结果 隐藏							
书号	类别	书名	出版社	出版年份	作者	价格	库存
5	计算机	数据库系统原理	机械工业	1997	Mark	109.99	68

可以发现，书号为 5 的图书的库存变化符合预期。修改库存前存量为 38，增加库存 30，修改后库存为 68。

进一步测试，修改该图书库存-67：



图书库存修改

书号: 5

变化量: -67

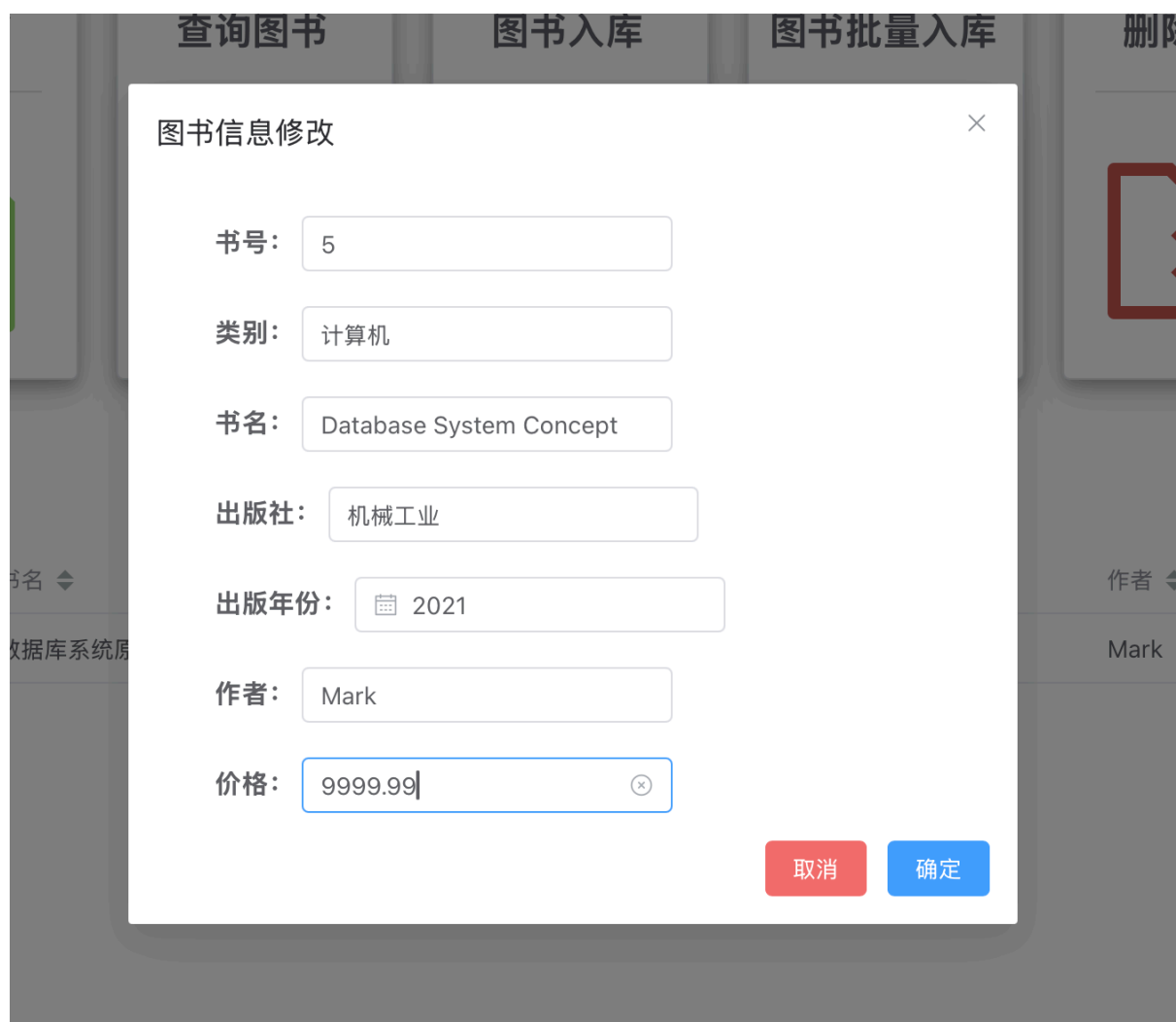
取消 确定

预期修改后库存为 $68-67=1$ ，查询图书观察测试结果，发现为 1，符合预期，测试成功：

图书查询结果 隐藏							
书号	类别	书名	出版社	出版年份	作者	价格	库存
5	计算机	数据库系统原理	机械工业	1997	Mark	109.99	1

5.2.1.8 图书信息修改

同理，点击“图书信息修改”按钮，在弹出的对话框中输入待修改的书号，以及需要修改的信息：



图书信息修改

书号： 5

类别： 计算机

书名： Database System Concept

出版社： 机械工业

出版年份： 2021

作者： Mark

价格： 9999.99

取消 确定

点击“确定”按钮，即可完成修改。

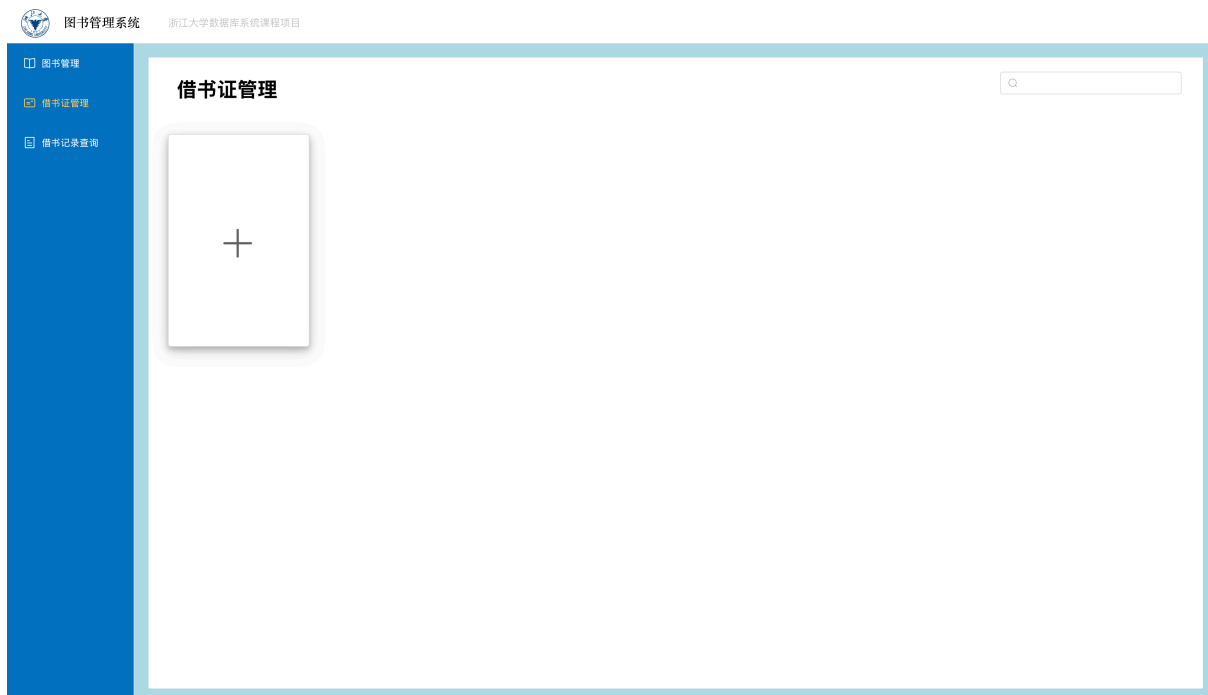
先后查询此图书以确认修改成功：



书号	类别	书名	出版社	出版年份	作者	价格	库存
5	计算机	Database System Concept	机械工业	2021	Mark	9999.99	1

可以看到，图书信息成功完成修改，一切功能正常。

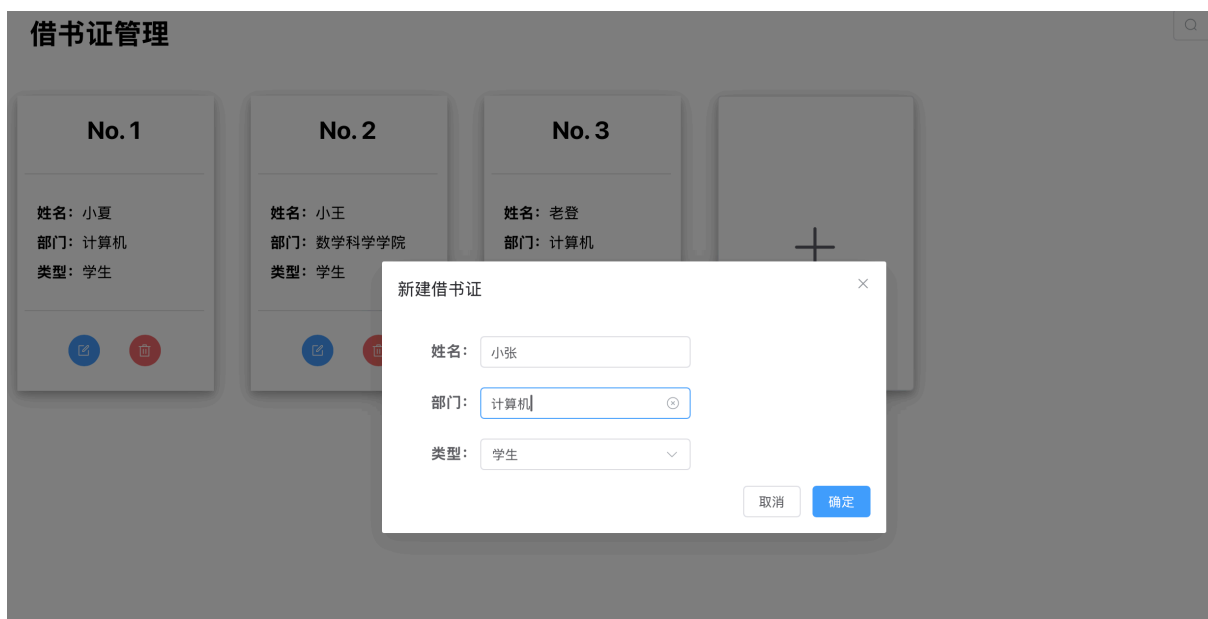
5.2.2 借书证管理页面



其中，查询借书证模块在页面渲染或信息修改时将自动调用，其结果可以直观查看，因此功能性测试主要围绕以下方面进行：新增借书证、修改借书证信息、删除借书证。

5.2.2.1 新增借书证

如图所示，借书证显示区域的前面部分显示已经注册的借书证，最后为新建借书证按钮。点击按钮，弹出“新建借书证”对话框，填写新借书证的信息，如图所示：



点击确定按钮，确认新建借书证，即可完成新增操作。在自动调用借书证查询方法之后，会显示“注册借书证成功”的弹窗，如图所示。

✓ 注册借书证成功

5.2.2.2 修改借书证信息

每一张借书证卡片底部都有两个按钮，分别对应编辑借书证卡片以及删除借书证卡片的弹窗。点击左下角的编辑按钮，会弹出修改借书证信息的弹窗：

修改信息(借书证ID: 4)

姓名: 小张

部门: 计算机科学与技术学院

类型: S

取消 确定

依次填写要修改的信息，例如图中修改 ID 为 4 的借书证部门为计算机科学与技术学院，点击确定，即可完成修改：

✓ 修改借书证信息成功

No. 2

姓名: 小王
部门: 数学科学学院
类型: 学生

No. 3

姓名: 老登
部门: 计算机
类型: 教师

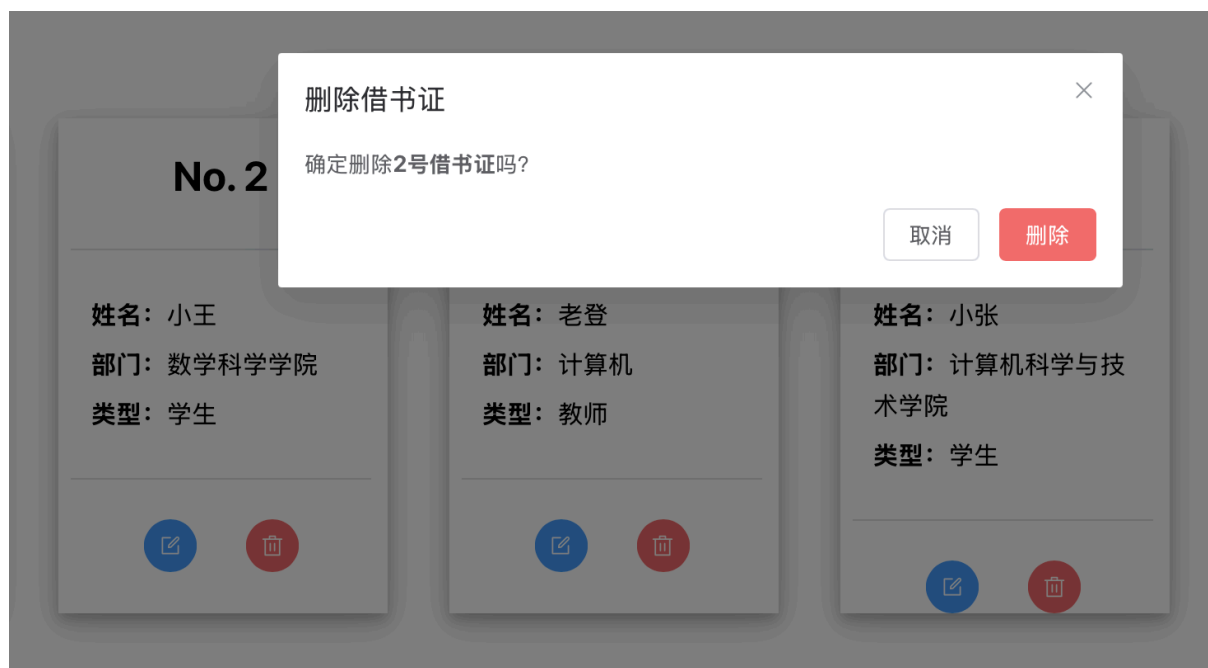
No. 4

姓名: 小张
部门: 计算机科学与技术学院
类型: 学生

如图所示，ID 为 4 的借书证卡片部门已修改为“计算机科学与技术学院”。

5.2.2.3 删除借书证

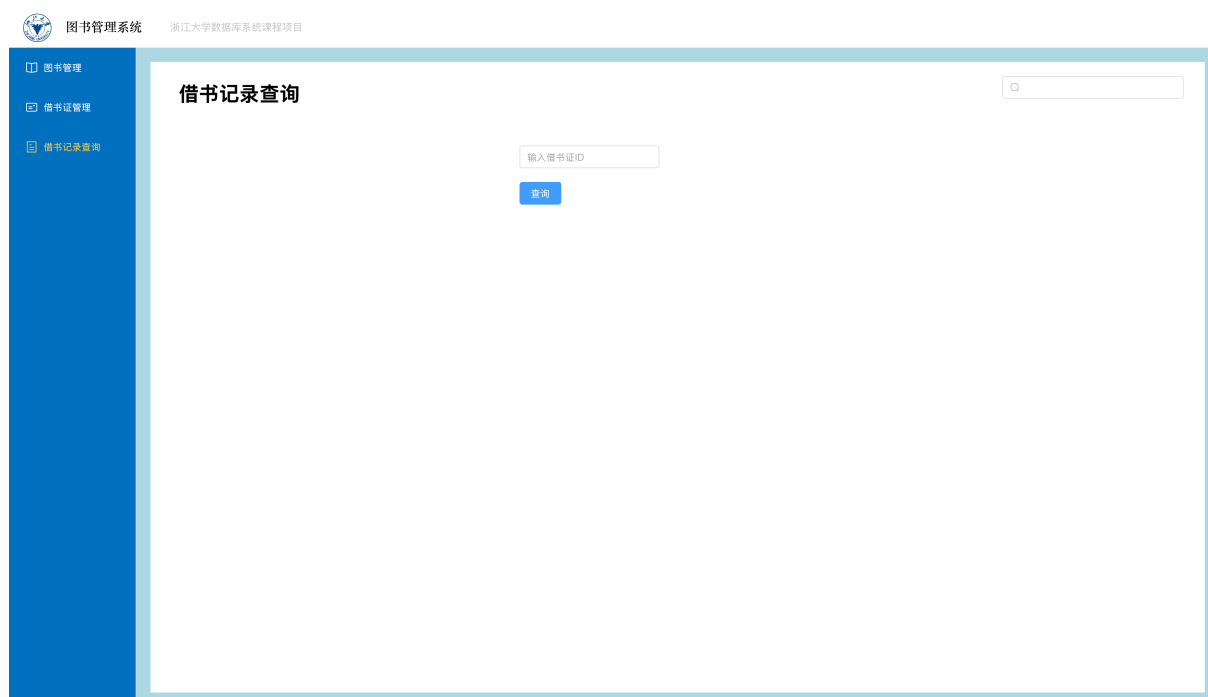
同理，点击要删除卡片右下角的删除按钮，会跳出一个对话框，询问是否确定删除：



点击确认即可删除：



5.2.3 借书记录查询页面



在上图“输入借书证 ID”对话框中输入要查询的借书证 ID，点击查询即可查询借书记录。下图示例查询借书证 ID 为 1 的借书记录，成功返回结果：

借书记录查询

<div>1</div> <div>查询</div>			
借书证ID	图书ID	借出时间	归还时间
1	2	2025年03月17日 00:00:00	2025年03月25日 11:03:02
1	3	2025年03月20日 09:28:14	未归还
1	4	2025年03月23日 03:01:01	未归还

6 问题与解决方案

6.1 前后端交互

6.1.1 出现的问题

在开发过程中，单独测试前端（npm run dev）或者后端都没有问题，但是前后端联调时出现问题，前端 console 报错，例如：

[Error] Failed to load resource: The network connection was lost.

[Error] Unhandled Promise Rejection: AxiosError: Network Error

而后端也会报错，例如：

The last packet sent successfully to the server was 0 milliseconds ago. The driver has not received any packets from the server.

```
at java.base/
jdk.internal.reflect.DirectConstructorHandleAccessor.newInstance(DirectConstructorHandleAccessor.java:62)
at java.base/java.lang.reflect.Constructor.newInstanceWithCaller(Constructor.java:501)
at java.base/java.lang.reflect.Constructor.newInstance(Constructor.java:485)
at com.mysql.cj.exceptions.ExceptionFactory.createException(ExceptionFactory.java:61)
at com.mysql.cj.exceptions.ExceptionFactory.createException(ExceptionFactory.java:105)
at com.mysql.cj.exceptions.ExceptionFactory.createException(ExceptionFactory.java:151)
at
com.mysql.cj.exceptions.ExceptionFactory.createCommunicationsException(ExceptionFactory.java:167)
at com.mysql.cj.protocol.a.NativeSocketConnection.connect(NativeSocketConnection.java:89)
at com.mysql.cj.NativeSession.connect(NativeSession.java:120)
at com.mysql.cj.jdbc.ConnectionImpl.connectOneTryOnly(ConnectionImpl.java:948)
at com.mysql.cj.jdbc.ConnectionImpl.createNewIO(ConnectionImpl.java:818)
... 7 more
Caused by: java.net.ConnectException: Connection refused
at java.base/sun.nio.ch.Net.connect0(Native Method)
at java.base/sun.nio.ch.Net.connect(Net.java:589)
at java.base/sun.nio.ch.Net.connect(Net.java:578)
at java.base/sun.nio.ch.NioSocketImpl.connect(NioSocketImpl.java:583)
at java.base/java.net.SocksSocketImpl.connect(SocksSocketImpl.java:327)
at java.base/java.net.Socket.connect(Socket.java:760)
at com.mysql.cj.protocol.StandardSocketFactory.connect(StandardSocketFactory.java:153)
at com.mysql.cj.protocol.a.NativeSocketConnection.connect(NativeSocketConnection.java:63)
... 10 more
Mar 25, 2025 6:49:57 PM Main main
```

6.1.2 解决方案

MySQL 默认监听 3306 端口，因此，后端端口必须设置成 3306，否则前后端就会连不上。同时，如果前后端端口不一致会引发跨域问题，所以需要在后端方法中加上处理跨域问题的代码，如下所示：

```
exchange.getResponseHeaders().add("Access-Control-Allow-Origin", "*");
exchange.getResponseHeaders().add("Access-Control-Allow-Methods", "GET, POST, OPTIONS");
exchange.getResponseHeaders().add("Access-Control-Allow-Headers", "Content-Type");
```

这样就能成功解决前后端连接失败的问题，保证前后端联调正常进行。

6.2 图书批量入库模块

6.2.1 出现的问题

与其他模块只需要一个对话框不同，图书批量入库模块需要能处理任意数量的入库书本。因此，像其他模块一样只出现一个对话框，里面包含一本图书的信息是不可行的，图书批量入库的模块需要能灵活处理多本书。

6.2.2 解决方案

在前端 Book.vue 中使用一个列表来存放多本图书的信息：

batchStoreBookInfo: []

在对话框中绑定两个按钮，“删除”与“增加”，分别对应删除一条图书信息与新增一条图书信息，并分别绑定两个事件，在点击时触发：

```
<el-table-column fixed="right" label="操作" style="width: 10%">
  <template #default="scope">
    <el-button link type="danger" size="small" @click.prevent="deleteRow(scope.$index)">
      删除
    </el-button>
  </template>
</el-table-column>
</el-table>
<el-button class="mt-4" style="width: 100%" @click="onAddItem">
  增加
</el-button>
```

两个方法如下所示，分别对应删除该行图书信息与新建一本书的信息：

```
deleteRow(index) {
  this.batchStoreBookInfo.splice(index, 1)
},
onAddItem() {
  this.batchStoreBookInfo.push({
    category: "",
    title: "",
    press: "",
    publishYear: "",
    author: "",
    price: "",
    stock: ""
  })
},
```

之后在 ConfirmBatchStore 事件被触发时，将列表中的每本书都作为一个 json 字符串加入空列表中，再将这个列表作为参数传递给后端：

```
this.batchStoreBookInfo.forEach(book => {
  book_list.push({
    bookId: '0',
    category: book.category,
    title: book.title,
    press: book.press,
    publishYear: book.publishYear,
    author: book.author,
    price: book.price,
    stock: book.stock
  })
})
```


6.3 日期格式

6.3.1 出现的问题

后端的数据定义中，borrowTime 与 returnTime 都是 long int 类型的数，即长整数。如果直接将长整数显示在前端界面中，会对用户造成很大困扰。同时，由于 0 代表图书未归还，而用户不理解 0 的具体含义，会认为是未定义的时间，因此也需要进行调整。

6.3.2 解决方案

在涉及显示借书时间与还书时间的前端代码 Borrow.vue 中，使用一个方法 转换时间为可理解的格式：

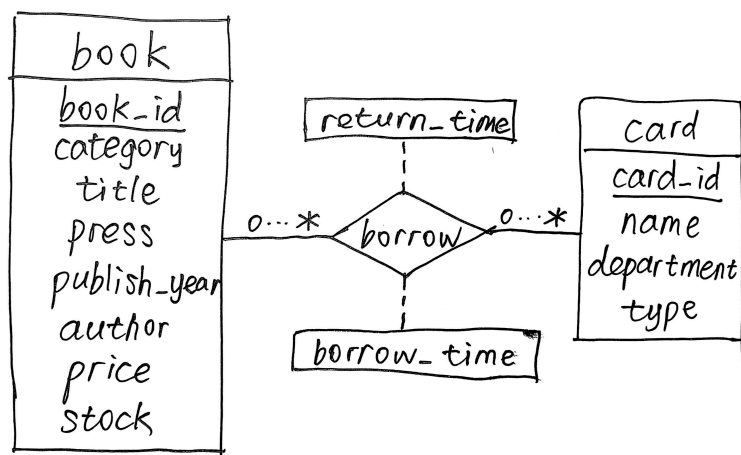
```
reflect(time) {  
  if (time === 0) {  
    return '未归还';  
  }  
  
  let t = time.toString();  
  let year = t.substring(0, 4);  
  let month = t.substring(4, 6);  
  let day = t.substring(6, 8);  
  let hour = t.substring(8, 10);  
  let minute = t.substring(10, 12);  
  let second = t.substring(12, 14);  
  return year + '年' + month + '月' + day + '日' + hour + ':' + minute + ':' + second;  
},
```

这样，就可以把 0 转换成“未归还”在界面中显示，把例如 20250304092613 格式的日期整数转换成 2025 年 03 月 04 日 09:26:13 这样的格式，更加用户友好。

7 思考题

7.1 绘制该图书管理系统的 E-R 图

如图所示：



7.2 描述 SQL 注入攻击的原理(并简要举例)。在图书管理系统中, 哪些模块可能会遭受 SQL 注入攻击? 如何解决?

7.2.1 SQL 注入攻击的原理

SQL 注入攻击是一种常见的网络安全攻击方式, 攻击者通过在输入字段中插入恶意的 SQL 代码, 试图篡改数据库查询语句, 从而获取、篡改或删除数据库中的数据。例如, 在登录系统中, 假设登录验证的 SQL 语句是:

```
SELECT * FROM users WHERE username = '$username' AND password = '$password';
```

如果用户输入的用户名是 admin' OR 1=1 --, 密码是任意值, 那么拼接后的 SQL 语句就会变成:

```
SELECT * FROM users WHERE username = 'admin' OR 1=1 --' AND password = '任意值';
```

由于 OR 1=1 始终为真, --是 SQL 的注释符号, 后面的语句被注释掉, 因此这条 SQL 语句会返回所有用户的信息, 攻击者可以借此绕过登录验证。

7.2.2 易受攻击的板块

- 用户登录模块: 如果用户名和密码的输入没有经过严格的过滤和验证, 攻击者可能会通过注入恶意 SQL 代码来获取管理员或其他用户的登录信息。
- 图书查询模块: 用户在查询图书时, 可能会输入恶意的 SQL 代码来篡改查询语句, 从而获取不该获取的图书信息或篡改图书库存等数据。
- 借书和还书模块: 在借书和还书操作中, 如果对用户输入的图书 ID、借书证 ID 等信息没有进行严格的验证和过滤, 攻击者可能会通过注入恶意 SQL 代码来篡改借书记录或还书记录, 从而导致图书库存不一致或借书记录混乱。

7.2.3 解决方案

1. 使用 PreparedStatement (项目中使用的)

PreparedStatement 是 Java 中的一种预编译 SQL 语句的技术, 它可以有效防止 SQL 注入攻击。通过使用 PreparedStatement, 可以将 SQL 语句中的参数与 SQL 语句本身分离, 从而避免了恶意 SQL 代码的注入。例如:

```
PreparedStatement stmt = conn.prepareStatement("SELECT * FROM users WHERE username = ? AND password = ?");  
stmt.setString(1, username);  
stmt.setString(2, password);  
ResultSet rs = stmt.executeQuery();
```

在这种情况下, 即使用户输入了恶意的 SQL 代码, PreparedStatement 也会将其视为普通的字符串参数, 而不会将其作为 SQL 语句的一部分来执行, 从而有效防止了 SQL 注入攻击。

2. 使用 ORM 框架

ORM (Object-Relational Mapping) 框架可以将数据库中的表映射为对象, 通过对象的操作来实现对数据库的操作, 从而避免了直接编写 SQL 语句。ORM 框架通常会自动处理 SQL 语句的拼接和参数绑定, 从而有效防止 SQL 注入攻击。

7.3 在 InnoDB 的默认隔离级别(RR, Repeated Read)下, 当出现并发访问时, 如何保证借书结果的正确性?

7.3.1 InnoDB RR

InnoDB 的默认隔离级别是 RR (Repeated Read), 在这种隔离级别下, 一个事务在读取数据时, 会看到该事务开始时数据库中数据的一致性视图。这意味着在同一个事务中, 即使其他事务对数据进行了修改, 该事务仍然会看到事务开始时的数据状态。这种隔离级别可以有效防止不可重复读 (Non-Repeatable Read) 问题, 即在同一个事务中, 对同一行数据的多次读取可能会得到不同的结果。

7.3.2 对应策略

1. 使用 synchronized 锁

在借书操作中, 可以使用 synchronized 锁来保证同一本书在同一时刻只能被一个用户借阅。例如:

```
synchronized (lock) {  
    // 检查库存  
    int stock = getStock(bookId);  
    if (stock <= 0) {  
        return new ApiResult(false, "图书库存不足! ");  
    }  
    // 更新库存  
    updateStock(bookId, stock - 1);  
    // 插入借书记录  
    insertBorrowRecord(borrow);  
}
```

在这种情况下, 通过 synchronized 锁, 可以确保同一本书在同一时刻只能被一个用户借阅, 从而避免了并发借书导致的库存不足和数据不一致问题。

2. 使用数据库事务

在借书操作中, 可以使用数据库事务来保证操作的原子性。例如:

```
try {  
    conn.setAutoCommit(false);  
    // 检查库存  
    int stock = getStock(bookId);  
    if (stock <= 0) {  
        conn.rollback();  
        return new ApiResult(false, "图书库存不足! ");  
    }  
    // 更新库存  
    updateStock(bookId, stock - 1);  
}
```

```
// 插入借书记录
insertBorrowRecord(borrow);
conn.commit();
} catch (Exception e) {
    conn.rollback();
    return new ApiResult(false, "Database Error:" + e.getMessage());
}
```

在这种情况下，通过数据库事务，可以确保借书操作的原子性。如果在借书过程中出现任何错误，事务会回滚，从而保证了借书记录和图书库存之间的一致性。

8 实验心得

这次实验是我第一次接触 web 前后端开发，也算是小小当了一次全栈工程师。看到自己写的网页最后成功跑起来，心里很有成就感。虽然实验过程中遇到了很多问题，一开始连项目结构都搞不清楚，对前后端 api 更是一无所知，但在完成项目的过程中也不断探索，学到了很多有用的技术：前端的 HTML, CSS, JavaScript, Vue, 后端的 Java, JDBC, 非常有收获。

个人感觉，最有挑战的是前后端 api 的部分。使用 GET, POST 的过程中需要传递 JSON 字符串，因此前端传给后端的、后端处理方法的对应变量名必须一致，否则 console 就会输出报错。

在这个项目之前，虽然我也写过不少代码，但往往都是一个或者几个文件，没有很复杂的结构，也没有涉及计算机网络相关的知识，都是在本地运行。通过这次实验，我极大提高了自己的项目开发能力，也基本掌握了 web 开发的有关内容，之后如果使用 docker 部署，就能成为一个真正可以运行的网页了。希望之后可以继续精进 web 开发的相关知识，把前端写的更美观，前后端 api 指定的更加优雅！