武汉大学计算机学院
本科生课程设计报告

移动编程技术大作业
图书商城 APP

专 业 名 称 ： 计算机科学与技术

课 程 名 称 ： 移动编程技术

指 导 教 师 ： 高建华

学 生 学 号 ： 2020302191970

学 生 姓 名 ： msm8976

二〇二二年五月

个人型设计报告学术声明示例：

郑 重 声 明

本人呈交的设计报告，是在指导老师的指导下，独立进行实验工作所取得的成果，所有数据、图片资料真实可靠。尽我所知，除文中已经注明引用的内容外，本设计报告不包含他人享有著作权的内容。对本设计报告做出贡献的其他个人和集体，均已在文中以明确的方式标明。本设计报告的知识产权归属于培养单位。

本人签名：msm8976　　日期：2022 年 5 月 23 日

目录

一、实验要求与目的

1.1 实验要求

设计图书商城 APP，实现的主要功能有：

1、Login 登录界面，可登录和注册，用户登录成功后将用户名写入 UserDefaults，下次登录无需再输入用户名。

2、主界面有三个 Tab，一个是图书列表，一个是购物车，一个是历史订单。

3、图书列表包括：

（1）可以浏览图书，按图书类别分区，区内展示本类别图书信息，每本图书有封面图标、书名、价格；

（2）图书详情：点击某书后，可以查看该书详细信息，包括该书封面大图，单价，书名、作者、书号和详情介绍；

（3）详情页可以将该书加入购物车；

（4）加入购物车后，在购物车页 Tab 上显示数量。

4、购物车页功能包括：

（1）所有选购图书的列表；

（2）可以对该列表中的图书进行删除；

（3）有“支付”按钮确认后显示一个支付成功页面，并将购物车的内容加入历史订单中。“取消”后返回上一级。

5、历史订单页功能包括：

（1）列出订单编号，图书数目，总价，购买时间；

（2）点击后展示该订单图书明细。

所有数据包括用户，图书，购物车、历史订单信息全部存储在 Sqlite 数据库。图书信息通过 web 接口（http://zy.whu.edu.cn/cs/api/book/list）获取图书列表。

1.2 实验目的

本次实验为移动编程技术课程的期末大作业，实验中大部分内容都在平时作业中完成过类似功能。在实验的过程中，需要我们复习巩固课堂上所学习的内容并将其整合起来，在 UI 设计、架构设计上相比平时作业也有了更高的要求。此外，对于实验中遇到的少部分新功能与困难，也需要进行解决、完善，锻炼了我们独立进行 iOS 应用开发的实践能力。

二、实验设计

2.1 实验环境

硬件：Intel Core i7-8550U 16+256

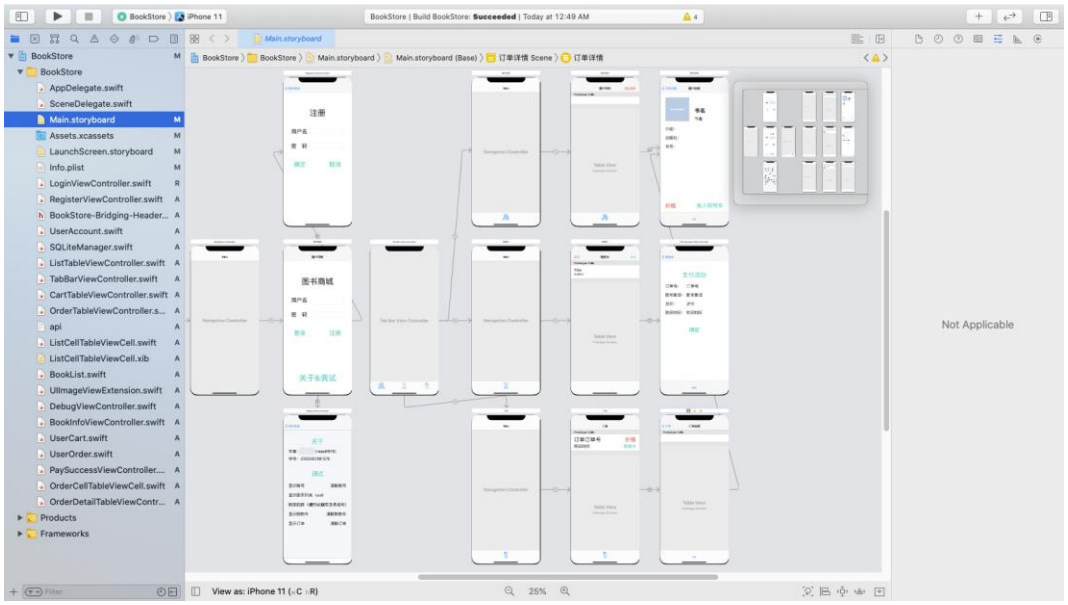
系统：黑苹果 macOS Catalina 10.15.7

版本：Xcode 12.4

模拟器：iPhone 11

2.2 流程架构设计

根据实验要求，设计如下的流程图与功能架构。



2.3 界面设计

2.3.1 总体设计

本次实验为所有界面内的控件均设置了合理的约束，对齐至以 16 为倍数的网格上。使用 iPhone 11 模拟器进行设计与开发，经测试在不同设备上均可正常显示，TableView 在横屏状态下也可正常显示。

2.3.2 启动界面

2020302191970
msm8976@qq.com

启动界面显示软件图标与名称，图标提取自 Pure 清羽图标包中为魅族读书设计的图标，软件中主题色也设置为该图标颜色（#00BFA5）

2.3.3 登录界面

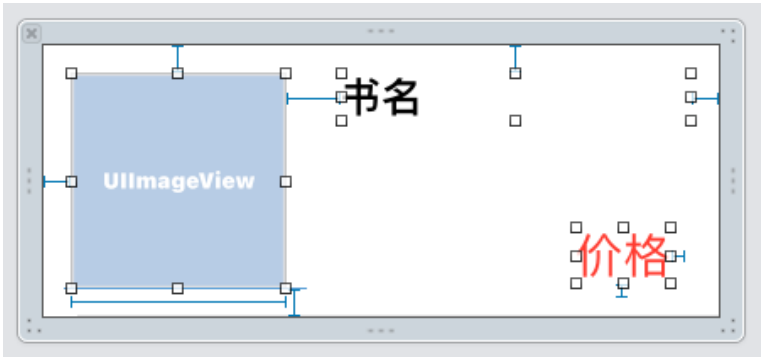
登录界面用户名读取上次登录写入的 UserDefaults，点击注册按钮可前往注册界面进行注册账号，也可点击关于&调试按钮进入调试界面，在登录成功后会将 rootViewController 设置为主界面的 TabBarViewController。

2.3.4 图书列表界面

图书列表界面使用自定义 xib 的 TableViewCell，通过 TableView 展示图书列表，点击图书可跳转至图书详情界面，在购物车不存在该商品时点击右下角按钮可将图书加入至购物车。

2.3.5 自定义 Cell

通过 api 查看图书图片均为正方形，因此设计如下的自定义 Cell。



2.3.6 购物车界面

购物车界面通过 TableView 展示当前用户的购物车，下方 TabBar 图标右上角显示购物车图书数量，左上角显示购物车内图书总价并随着购物车修改而实时刷新，右上角支付按钮点击后显示确认支付弹窗，确定后跳转至支付成功界面，并显示订单信息。

2.3.7 订单界面

订单界面使用 Custom Cell 的 TableView 展示当前用户的订单，点击订单后跳转至订单详情界面，以订单内包含的图书初始化复用图书列表的 TableView，点击图书也可跳转至图书详情，相较第一个界面不显示加入购物车按钮。

2.4 数据库设计

2.4.1 UserAccount

UserAccount 存储用户账号信息，设置主键为'username' TEXT NOT NULL，外键为'password' TEXT

2.4.2 Books

Books 存储图书信息，每次登录时会判断 api 是否正常返回图书信息，若正常则清空数据库并将新的图书数据写入数据库，设置主键为 'id' TEXT NOT NULL，其余信息设置为 api 对应 key 名称的 TEXT 外键。

2.4.3 UserCart

UserCart 存储用户的购物车信息，由于每个用户购物车中可有多本书，不同用户购物车中可有相同的书，因此设置主键为'id' INTEGER NOT NULL AUTOINCREMENT，外键为'username' TEXT , 'bookid' TEXT。

2.4.4 UserOrder

UserCart 存储用户的订单信息，由于本次实验仅为本地测试不考虑并发，将订单号设置为秒级时间戳，因此订单时间可通过订单号计算出来。设置主键为'orderid' TEXT NOT NULL，外键为'username' TEXT , 外键为'username' TEXT , 'bookls' TEXT , 'price' TEXT , 'num' TEXT，分别表示用户名、图书 id 列表、订单总价、订单图书数量。

三、实验重要代码

3.1 数据库相关

数据库的创建、查询、修改、删除与平常作业相似，不再介绍。

以下为图书信息数据库初始化逻辑，判断 api 是否正常写入图书信息列表，若信息列表不为空则清空数据库并将新的图书数据写入数据库。

```
static func initDB(){
    let sqlite = SQLiteManager.sharedInstance
    if !sqlite.openDB() {return}
    let createSql = "CREATE TABLE IF NOT EXISTS Books('id' TEXT NOT NULL
PRIMARY KEY , 'kind' TEXT , 'title' TEXT , 'author' TEXT , 'publisher'
TEXT , 'code' TEXT , 'price' TEXT , 'description' TEXT , 'pic' TEXT);"

if !sqlite.execNoneQuerySQL(sql:createSql){sqlite.closeDB();return}
    if booklist.count==0{
        print("api 获取失败，使用本地数据库")
        let queryResult = sqlite.execQuerySQL(sql:"SELECT * FROM Books;")
        for qr in queryResult!{
            let data: Data = try! JSONSerialization.data(withJSONObject:
qr, options: .fragmentsAllowed)
            let tempbookinfo=try! JSONDecoder().decode(bookInfo.self,
from:data)
            booklist.append(tempbookinfo)
        }
        sqlite.closeDB()
        return
    }
    let clean = "DELETE FROM Books"
    if !sqlite.execNoneQuerySQL(sql:clean){sqlite.closeDB();return}
    var insertsql=""
    for bookinfo in booklist{
        insertsql = "INSERT INTO
Books(id,kind,title,author,publisher,code,price,description,pic)
VALUES('"+bookinfo.id+"','"+bookinfo.kind+"','"+bookinfo.title+"','"+
bookinfo.author+"','"+bookinfo.publisher+"','"+bookinfo.code+"','"+bo
okinfo.price+"','"+bookinfo.description+"','"+bookinfo.pic+"');"
```

```
if !sqlite.execNoneQuerySQL(sql:insertsql){sqlite.closeDB();return}
    }
    sqlite.closeDB()
}
```

为了便于查看数据库信息，本次实验为三个用户数据库设计了查询、清空、删除函数，便于调试使用，以下代码为每个数据库各显示一个函数。

```
static func clean() -> Bool
{
    let sqlite = SQLiteManager.sharedInstance
    if !sqlite.openDB() {return false}
    let clean = "DELETE FROM UserAccount"
    if !sqlite.execNoneQuerySQL(sql:clean){sqlite.closeDB();return
false}
    sqlite.closeDB()
    return true
}

static func show() -> [[String : AnyObject]]?
{
    let sqlite = SQLiteManager.sharedInstance
    if !sqlite.openDB(){return nil}
    let queryResult = sqlite.execQuerySQL(sql:"SELECT * FROM UserCart;")
    sqlite.closeDB()
    return queryResult!
}

static func drop(){
    let sqlite = SQLiteManager.sharedInstance
    if !sqlite.openDB() {return}
    let drop = "DROP TABLE UserOrder;"
    if !sqlite.execNoneQuerySQL(sql:drop){sqlite.closeDB();return}
    sqlite.closeDB()
    return
}
```

3.2 登录逻辑

登录界面加载时初始用户账号数据库，显示用户名的

UserDefaults，并通过 api 获取图书数据。点击登录按钮后判断用户是否存在，若用户密码匹配则初始化图书列表、购物车、订单数据库，将 rootViewController 设置为主界面的 TabBarViewController。

```
override func viewDidLoad() {
    super.viewDidLoad()
    UserAccount.initDB()
    loginUserInput.text=defaults.string(forKey: "username")
    bookList.initBookList()
}

@IBAction func loginTapped(_ sender: Any) {
    if !UserAccount.isUserExist(username: loginUserInput.text!){
        loginTint.text = "用户不存在"
        loginTint.textColor = .red
        defaults.set("",forKey: "username")
    }else if UserAccount.login(username: loginUserInput.text!, password:
loginPwdInput.text!){
        loginTint.text = "登录成功"
        loginTint.textColor = .green
        UserAccount.user=loginUserInput.text!
        bookList.initDB()
        UserCart.initDB()
        UserOrder.initDB()
        defaults.set(loginUserInput.text!,forKey: "username")
        let tabvc =
storyboard?.instantiateViewController(withIdentifier: "tab") as!
UITabBarController
        self.view.window?.rootViewController = tabvc
    }else{
        loginTint.text = "用户名或密码错误"
        loginTint.textColor = .red
        defaults.set(loginUserInput.text!,forKey: "username")
    }
}
```

3.3 注册逻辑

用户注册时判断用户名是否与已有用户相同，若不同则将用户名写入 UserDefaults 并返回登录界面。

```
@IBAction func regOKTapped(_ sender: Any) {
    if UserAccount.isUserExist(username: regUserInput.text!){
        regTint.text = "用户名已存在"
        regTint.textColor = .red
    }else if UserAccount.register(username: regUserInput.text!, password:
regPwdInput.text!){
        regTint.text = "注册成功"
        regTint.textColor = .green
        defaults.set(regUserInput.text!,forKey: "username")
        self.navigationController?.popViewController(animated: true)
    }else{
        regTint.text = "注册失败"
        regTint.textColor = .red
    }
}
```

3.4 图书列表初始化

由于图书列表需按照分类设置 section，因此根据图书 list 的种类建立字典，并将图书信息加入字典对应种类的 list 中。

```
override func viewDidLoad() {
    super.viewDidLoad()
    for bookinfo in bookList.booklist{
        if bookdict[bookinfo.kind] == nil{
            bookkind.append(bookinfo.kind)
            bookdict[bookinfo.kind]=[]
        }
        bookdict[bookinfo.kind]?.append(bookinfo)
    }
    let xib = UINib(nibName: "ListCellTableViewCell", bundle: nil)
    tableView.register(xib, forCellReuseIdentifier: "cell1")
    tableView.rowHeight = 160
    self.tableView.sectionIndexColor = themeColor
}
```

3.5 提交订单

实验要求点击提交订单后弹出确认弹窗，点击确认后显示支付成功界面，不便于直接给按钮绑定 segue，因此将 shouldPerformSegue() 设置为 false 并在弹窗中手动调用 performSegue。

```
@IBAction func payTapped(_ sender: Any) {
    if idlist.count != 0{
        let dialog = UIAlertController(title:"提示", message:"是否确认支付本订单", preferredStyle: .alert)
        dialog.addAction(UIAlertAction(title: "取消", style: .cancel, handler: nil))
        dialog.addAction(UIAlertAction(title: "支付", style: .default, handler: {(act:UIAlertAction) in self.performSegue(withIdentifier: "pay", sender: self)}))
        present(dialog, animated:true, completion:nil)
    }else{
        let dialog = UIAlertController(title:"提示", message:"购物车中不存在商品", preferredStyle: .alert)
        dialog.addAction(UIAlertAction(title: "确定", style: .default, handler: nil))
        present(dialog, animated:true, completion:nil)
    }
}

override func shouldPerformSegue(withIdentifier identifier: String, sender: Any?) -> Bool {
    return false
}
```

3.6 订单详细信息

点进订单详情需要获取当前订单所包含的图书的信息，使用 filter 闭包实现判断。

```
override func viewDidLoad() {
    super.viewDidLoad()
    orderbooklist=bookList.booklist.filter({
        for id in orderbookidlist{
            if $0.id == id{
                return true
            }
        }
        return false
    })
    let xib = UINib(nibName: "ListCellTableViewCell", bundle: nil)
    tableView.register(xib, forCellReuseIdentifier: "cell4")
    tableView.rowHeight = 160
}
```

3.7 界面复用与数据传递

本次图书列表-订单详情、图书详细-订单图书详情使用的时相同的自定义 Cell 与 ViewController，其中图书列表 Cell 完全相同，而通过图书列表进入的图书详情界面应显示加入购物车按钮，而订单图书详情界面不应显示加入购物车按钮，通过 prepare 函数为 hideAddCart 变量设置不同的 Bool 来实现。本次实验由于不涉及到子界面修改数据，仅包含数据的传出，无需考虑数据的传回，因此未使用 prepareForUnwind()函数。

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    guard let bookinfovc=segue.destination as? BookInfoViewController
else {return}
    let indexPath=tableView.indexPathForSelectedRow

bookinfovc.bookinfo=bookdict[bookkind[indexPath!.section]]![indexPath
!.row]
    bookinfovc.hideAddCart=false
}

override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    guard let bookinfovc=segue.destination as? BookInfoViewController
else {return}
    let indexPath=tableView.indexPathForSelectedRow
    bookinfovc.bookinfo=orderbooklist[indexPath!.row]
    bookinfovc.hideAddCart=true
}
```

四、实验流程演示

4.1 注册登录

启动后默认显示登录界面，用户名为上次登录写入的UserDefaults，输入正确的用户名密码可登录，否则提示不同的错误信息，登录成功后会将根界面切换至内容界面。

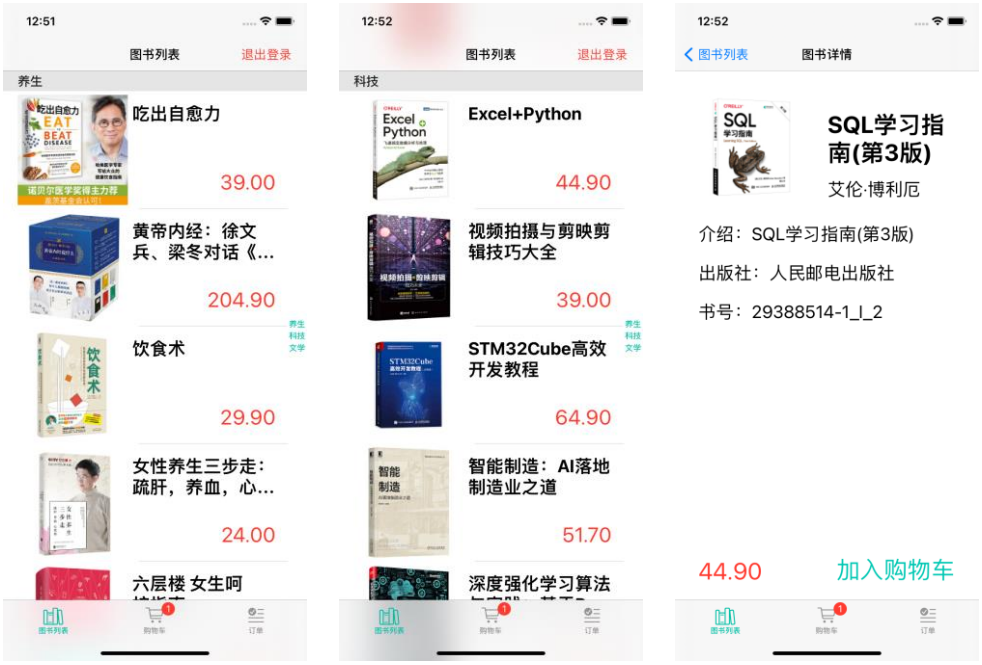
注册界面会判断用户名是否存在，若不存在可正常注册。

4.2 关于调试

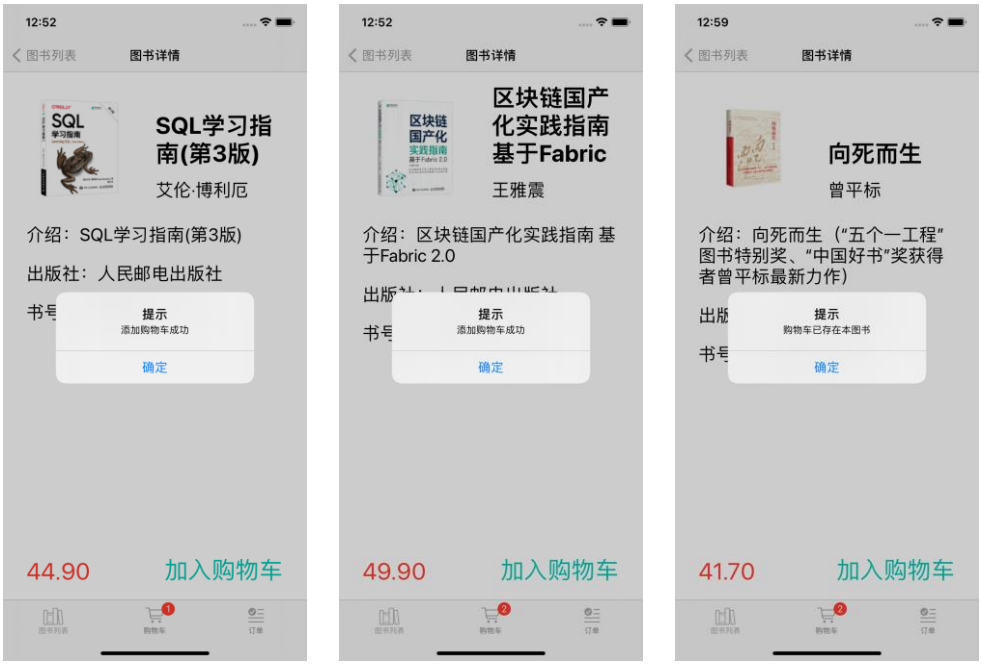
调试界面可点击按钮在控制台显示数据库信息或执行清除操作。

4.3 加入购物车

图书列表可上下滑动，点击右侧文字可以定位到具体的 section，点击图书可查看图书详情。

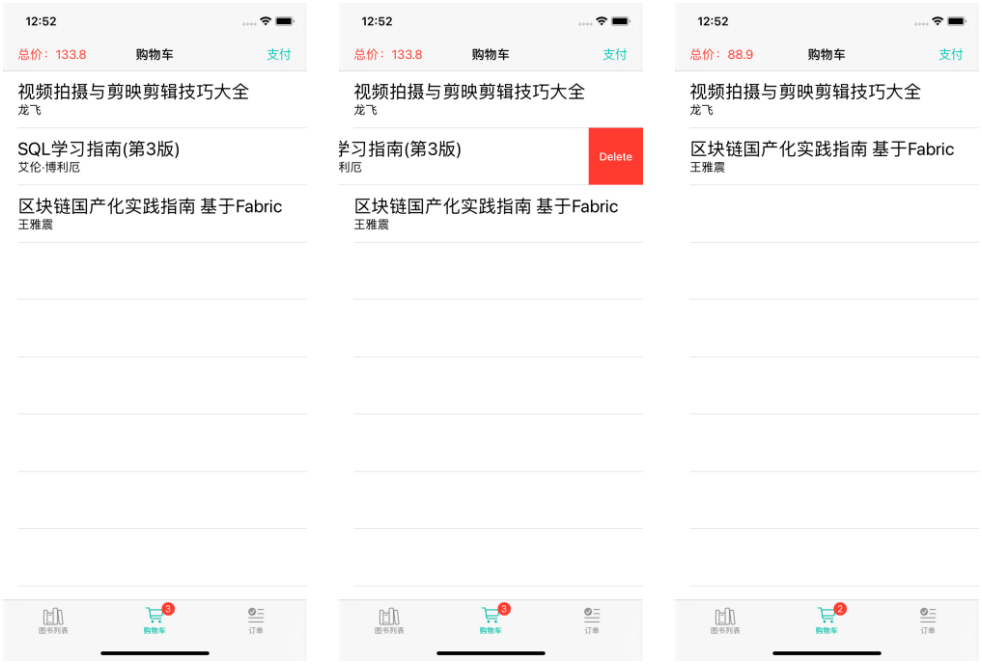


点击右下角加入购物车按钮可将图书加入购物车并返回图书列表界面，加入成功后购物车右上角角标会对应变化，若购物车已存在该图书也会弹窗提示。



4.4 修改并提交订单

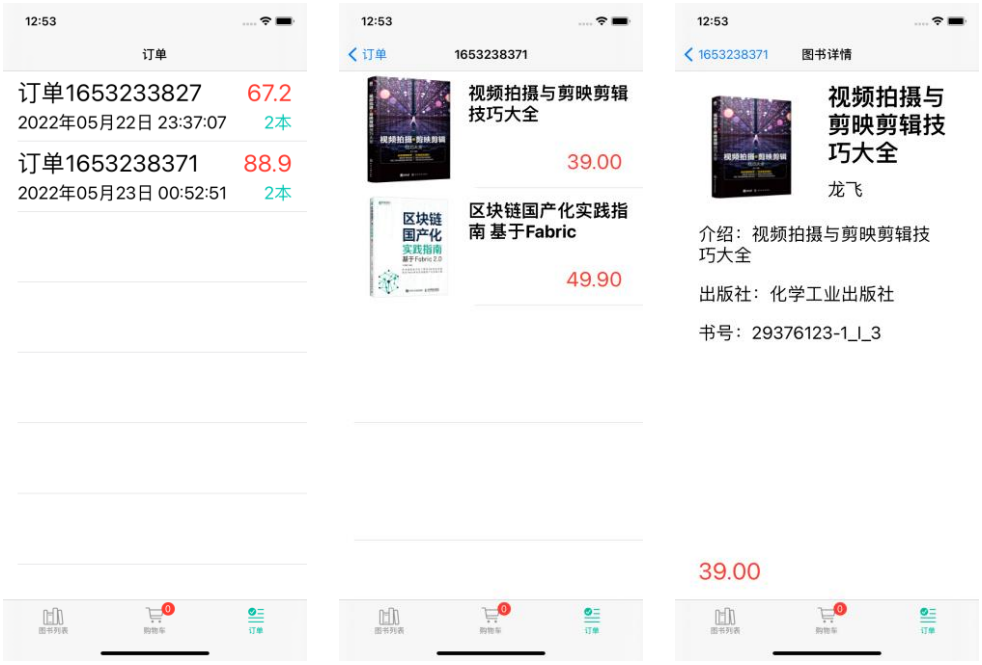
TabBar 点击购物车查看当前购物车，左上角显示当前购物车图书总价，左滑可以删除购物车，购物车图书总价会随之更新。



点击右上角支付按钮会弹出支支付确认弹窗，点击支付后进入支付成功界面，显示该订单信息，返回后购物车已清空。

4.5 查看订单

TabBar 点击订单查看历史订单，点击订单可查看订单所包含的图书，点击图书可查看图书详情。



4.6 切换账户

首页右上角点击退出登录按钮可跳转至登录界面登录其他账号，不同账号的购物车、订单信息相互独立互不影响，图片为另一个账号的购物车及订单信息。

五、难点与解决方式

5.1 登录跳转

登录后的界面与登录完全无关，因此登录时将根界面切换至内容的 TabBar，以免包含返回等无关按钮。

```
let tabvc = storyboard?.instantiateViewController(withIdentifier: "tab")
as! UITabBarController
self.view.window?.rootViewController = tabvc
```

5.2 主题色设置

Label、按钮可以通过右侧工具栏设置颜色，而 TabBar、section 右侧按钮无法直接设置，需要使用 RGB 色值初始化自定义颜色，并通过代码将其设置为该自定义颜色。

```
let themeColor = UIColor.init(red: (0 / 255.0), green: (191 / 255.0), blue:
(165 / 255.0), alpha: 1);
override func viewDidLoad() {
    super.viewDidLoad()
    self.tabBar.tintColor = themeColor
}
```

5.3 自定义 Cell 点击

图书列表界面建立包含 xib 的自定义 Cell 后，通过按住 CTRL+鼠标拖拽建立的 segue 无法做到点击跳转，因此改用 didSelectRowAt 函数进行跳转。

```
override func tableView(_ tableView: UITableView, didSelectRowAt
indexPath: IndexPath) {
    performSegue(withIdentifier: "showbookinfo", sender: self)
}
```

5.4 在线图片刷新

使用图片 extens 的 downloadAsyncFrom()在线图片加载函数加载

图书图片时，由于图片为异步下载，TableView 中首次显示图片为空，需要切换界面或重新加载后才可正常显示。因此，初始化 needReload 的 Bool 变量为 true，在首次加载到最后一个 cell 时执行 reloadData() 函数刷新，保证图片的正常显示，订单详情界面也进行相同的处理。

```
override func tableView(_ tableView: UITableView, cellForRowAt indexPath:
IndexPath) -> UITableViewCell {
    let cell = tableView.dequeueReusableCell(withIdentifier: "cell1",
for: indexPath) as! ListCellTableViewCell
    let booklist=bookdict[bookkind[indexPath.section]]!
    cell.imageView!.downloadAsyncFrom(url:
booklist[indexPath.row].pic)
    cell.cellName.text=booklist[indexPath.row].title
    cell.cellPrice.text=booklist[indexPath.row].price
    if indexPath.row==booklist.count-1 && needReload{
        needReload=false
        tableView.reloadData()
    }
    return cell
}
```

5.5 购物车角标

购物车数量角标需要在增加、删除购物车后改变，增加购物车在图书列表界面完成，因此需要在 Tab 切换至购物车界面时刷新，删除购物车在购物车界面滑动，因此需要在删除 cell 后刷新（本次实验直接调用上方写好的 viewWillAppear()函数。

```
override func viewWillAppear(_ animated: Bool) {
    sumprice=0
    idlist=UserCart.getCart()
    for id in idlist{
        infodict[id]=bookList.getBookInfoById(id: id) as?
[String:String]
    }
    tableView.reloadData()

self.navigationController?.tabBarItem.badgeValue=String(idlist.count)
```

```
    DispatchQueue.main.asyncAfter(deadline: .now()+0.02) {
        self.priceBtn.title="总价:
"+String(Float(lroundf(10*self.sumprice))/10.0)
    }
}

override func tableView(_ tableView: UITableView, commit editingStyle:
UITableViewCell.EditingStyle, forRowAt indexPath: IndexPath) {
    if editingStyle == .delete {
        if UserCart.delCart(bookid:idlist[indexPath.row]){
            tableView.deleteRows(at: [indexPath], with: .fade)
            viewWillAppear(true)
        }
    }
}
```

5.6 订单总价

订单总价若在支付订单时计算较为简单，但由于在购物车界面左上角实时显示订单价格，新增、删除购物车后需要重新计算，因此使用和 5.4 相同的刷新事件。计算总价时需要遍历购物车中每一本图书的价格，因此直接在加载 cell 时在 sumprice 加上所对应图书的价格，并在需刷新 TableView 将 sumprice 清零。但在 viewWillAppear()内若直接输出 sumprice 会因为 cell 还未加载完无法获取正确的总价，因此使用异步加载，延迟 0.02s 后再显示总价，此延迟操作在使用 APP 时无感知。

```
override func tableView(_ tableView: UITableView, cellForRowAt indexPath:
IndexPath) -> UITableViewCell {
    let cell = tableView.dequeueReusableCell(withIdentifier: "cell2",
for: indexPath)
    cell.textLabel?.text=infodict[idlist[indexPath.row]]?["title"]

cell.detailTextLabel?.text=infodict[idlist[indexPath.row]]?["author"]
    sumprice+=(Float((infodict[idlist[indexPath.row]]?["price"])!)!)
    return cell
}
```

六、实验总结与感想

本次移动编程技术课程相比于大多数课程，更加偏向于实践。在每次的课后作业中，高老师都会发送对应知识点的详细操作视频供我们参考，而本次大作业也是对之前知识点的回顾与总结，因此对于不太清楚的功能可以参考视频来实现。

课程之初最难解决的是环境的配置，安装虚拟机、下载镜像、选择 macOS 与 XCode 版本都耗费了不少时间与精力，且即使给虚拟机分配 8 核 8GB 内存仍会因为无法调用核显而在显示动画时卡顿。最终选择了购买一台可以安装黑苹果的笔记本，通过 Clover 引导安装了最后一个仅适配 X86 的 macOS10.15.7 系统与对应系统版本的 XCode12.4，在后续的作业与实验中都可以流畅的使用。

相比平时作业，本次大作业由于没有步骤教程，最为重要的其实是开始的实验设计，在还未实现具体功能前先对程序进行一个整体的初步设想，考虑程序需要哪些界面与每个界面的 UI、功能。在最开始时登录界面跳转注册使用的 Present Modally，后期考虑到统一设计全部改用 Navgation Controller 的 show。为了兼容不同大小的设备，本次实验对所有的控件都设置了顶部/底部、左侧/右侧至少两条约束，或进行水平/竖直居中，并对齐至以 16px 为倍数的网格上，使程序界面规整美观。此外，数据库表结构由于开始没有完整考虑使用场景，其设计也经过了数次修改，为此加入了调试界面便于查看和删除数据库信息。

功能代码的编写上没有遇到大的困难，基本都是参考了平时作业

的思路与解决方式，但也在测试中发现了一些 bug，例如在线图片在 cell 中首次加载不显示、购物车数量与总价无法实时更新等，虽然可能不是最佳修复方式，这些 bug 都通过自己的方式进行了解决，目前提交的版本经本人测试未发现 bug。

在程序基本完成后，也发现了早期部分的设计的缺陷，例如 api 获取图书信息与存入数据库相对孤立，程序大部分位置都使用的 api 初始化的图书列表而不是根据筛选条件读取数据库，为此特意在初始化数据库时增加了 api 数据判断，若因网络原因获取失败则使用历史数据库来初始化的图书列表；为了方便将订单图书列表以空格分隔生成的字符串存入数据库而不是转为 json，无法兼容未来可能存在的包含空格的图书 id。此类缺陷在今后程序开发的初始设计时就应该想办法避免，以免在后续造成更大工作量的修改与影响。

总之，这次实验是我首次独立的设计开发一个移动软件，完整的经历了设计、实现、测试、修复的过程，提高了我的实践能力，也让我对程序开发有了更深入的理解。在今后的程序开发中，我也会吸取本次的经验，避免再次出现相同的问题。

最后，感谢高老师一学期以来的辛勤付出，每节课上课前都将相应的资料与操作视频传到群里，上课时在活跃的气氛下讲授课程，课后在群里耐心解答同学们的问题，也希望今后我能在 python、移动编程技术外再次选修高老师讲授的课程。

教师评语评分

评语：

评分：

评阅人：

年　　月　　日

（备注：对该实验报告给予优点和不足的评价，并给出百分制评分。）