文档Lab06-详细设计文档

文档作者

主要编写者：史创屹、杨枫、刘存玺

文档修改历史

修改人员	日期	修改原因	版本号
史创屹	2024-5-15	完成初版	
杨枫	2024-5-19	修改业务逻辑层的分解	
刘存玺	2024-5-19	修改展示层的分解	

1. 引言

1.1 编制目的

本报告详细完成对 **蓝鲸商店系统** 的详细设计，达到指导后续软件构造的目的，同时实现和测试人员及用户的沟通。

本报告面向开发人员、测试人员及最终用户而编写，是了解系统的导航。

1.2 词汇表

详细设计描述的约定详见下表所示：

词汇名称	对应全称或具体含义
axios	HTTP网络请求库

1.3 参考资料

- 丁二玉 刘钦《软件工程与计算 卷2 软件开发的技术基础 》
- 用例文档
- 需求规格说明文档
- 体系结构设计文档

2. 产品概述

参考蓝鲸商店系统用例文档和蓝鲸商店管理系统软件需求规格说明文档中对产品的概括描述。

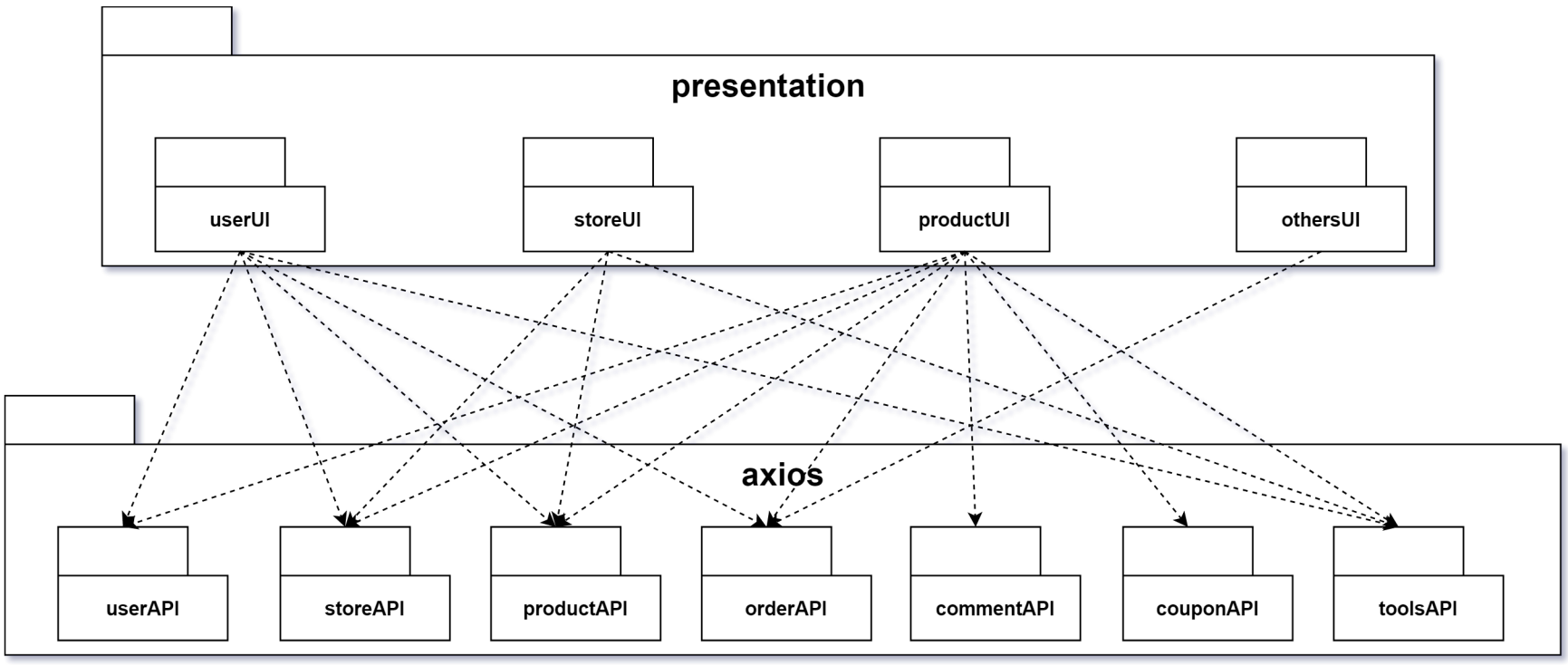
3. 体系结构设计概述

参考蓝鲸商店管理系统体系结构设计文档中对体系结构设计的概述。

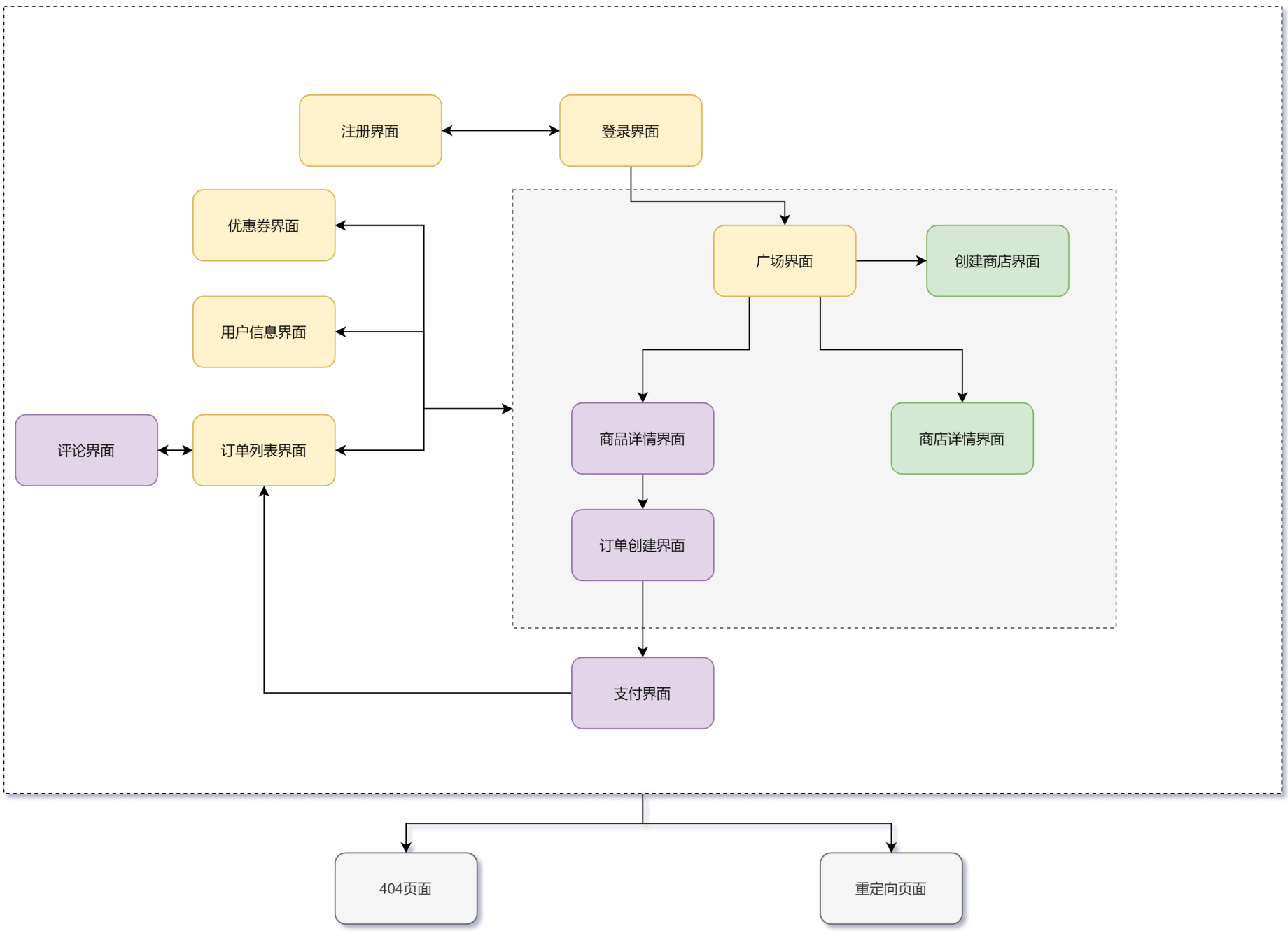
4. 结构视角

4.1 展示层的分解

展示层的开发包图



用户页面跳转图



4.1.1 userUI

以userUI为例

4.1.1.1 模块概述

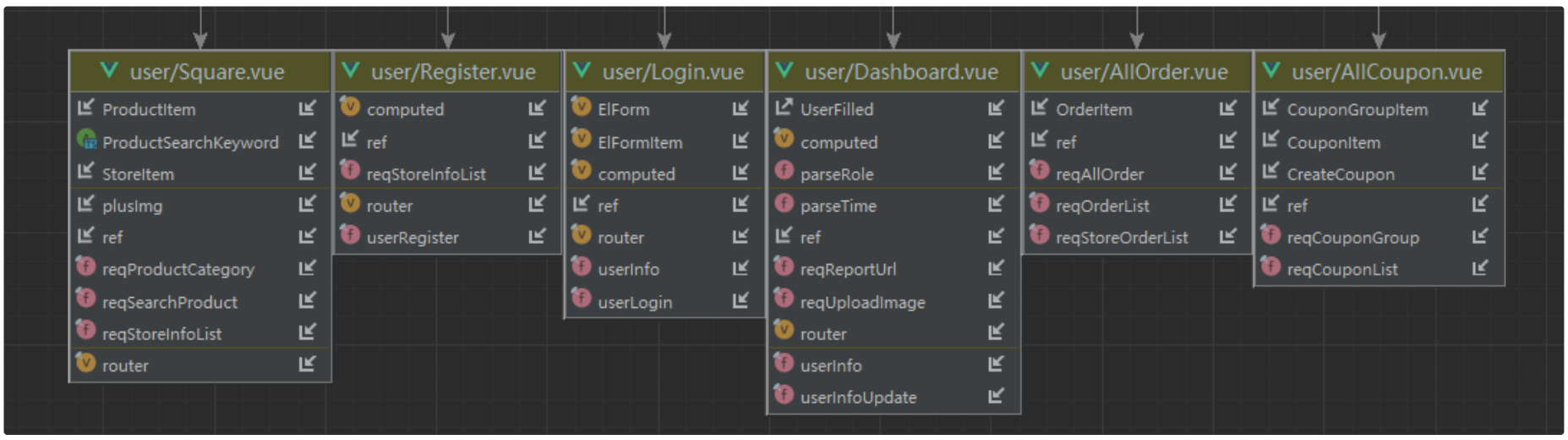
userUI包括展示层中和用户相关的界面。负责处理与用户信息高度相关的业务与发送相关网络请求。

4.1.1.2 整体结构

- 包括页面跳转图中的相关页面(橙黄色)
- userUI会调用相关网络API包向后端发送HTTP请求

视图类列表

名称	路径
登录界面	/src/views/user/Login.vue
注册界面	/src/views/user/Register.vue
广场界面	/src/views/user/Square.vue
用户信息界面	/src/views/user/Dashboard.vue
订单列表界面	/src/views/user/AllOrder.vue
优惠券界面	/src/views/user/AllCoupon.vue



4.1.1.3 模块内部类的接口规范

src/api/**user.ts**

名称	请求方法	接口参数	说明
userInfo	GET		请求用户信息
userInfoUpdate	POST	{name?: string, password?: string, address?: string,}	更新用户信息
userLogin	POST	{ phone: string, password: string}	登录
userRegister	POST	{role: string,name: string,phone: string,inviteCode?: number,password: string,address: string, storeId?: number}	注册

src/api/**coupon.ts**

名称	请求方法	接口参数	说明
reqCouponGroup	GET	{storeId: number}	请求优惠券组
reqCouponGroupCreate	POST	{id?: number,type: string, count: number,storeId?: number,threshold?: number,reduction?: number,inventory?: number,received?: boolean}	请求创建优惠券组
reqCouponList	GET		请求优惠券列表
reqCouponReceive	POST	{groupId: number}	请求领取优惠券
reqCouponCalculate	GET	{couponId: number, price: number}	计算优惠价格

src/api/**order.ts**

名称	请求方法	接口参数	说明
reqOrder	GET	{orderId: number}	请求订单
reqAllOrder	GET		请求所有订单
reqOrderList	GET		请求订单列表(根据用户id)
reqStoreOrderList	GET		请求订单列表(根据商店id)
reqOrderAdd	POST	{id: number,itemId: number,userId: number,status: string,delivery: string,storeId: number,couponIds: number[],price: number,quantity: number}	请求创建订单
reqOrderPay	POST	{traceNo: string,totalAmount: number,subject: string,alipayTradeNo: string}	请求支付订单
reqOrderSend	POST	{orderId: number}	请求发货
reqOrderReceive	POST	{orderId: number}	请求收货
reqOrderComment	POST	{orderId: number}	请求评论

src/api/**tools.ts**

名称	请求方法	接口参数	说明
reqUploadImage	POST		上传图片文件
reqImageList	GET		请求图片列表
reqReportUrl	GET		请求报表

4.1.2 storeUI

4.1.2.1 模块概述

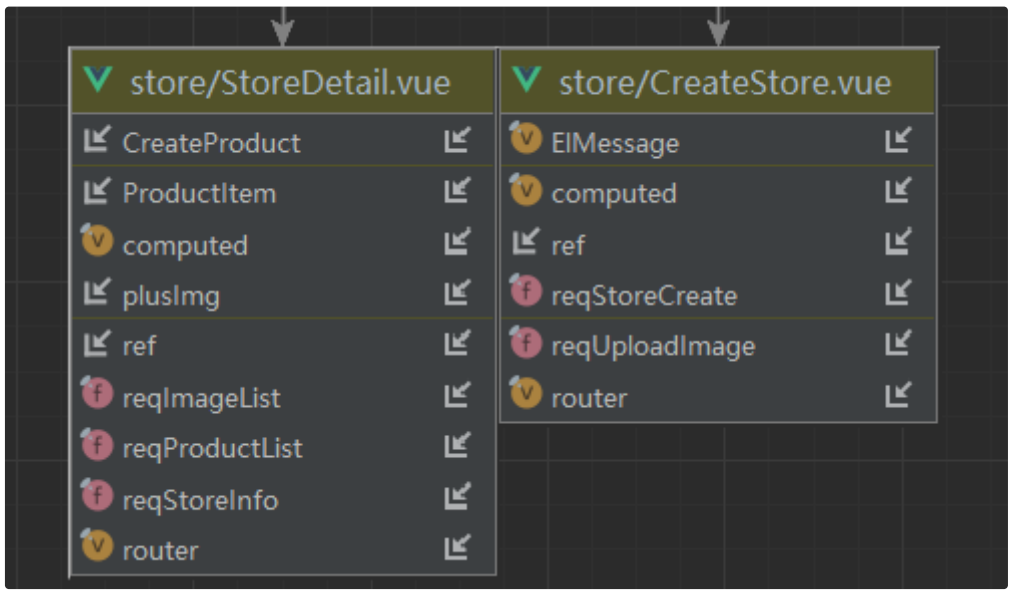
storeUI包括了和商店创建与访问相关的页面以及实现页面所需要的业务逻辑和网络请求。

4.1.2.2 整体结构

- 包括页面跳转图中的相关页面(浅绿色)
- storeUI会调用相关网络API包向后端发送HTTP请求

视图类列表

名称	路径
创建商店界面	/src/views/store/CreateStore.vue
商店详情界面	/src/views/store/StoreDetail.vue



4.1.2.3 模块内部类的接口规范

src/api/**store.ts**

名称	请求方法	接口参数	说明
reqStoreCreate	POST	{name: string,description: string}	创建商店
reqStoreInfo	GET	{storeId: number}	由商店ID请求商店信息
reqStoreInfoList	GET		获取商店列表

4.1.3 productUI

4.1.1.1 模块概述

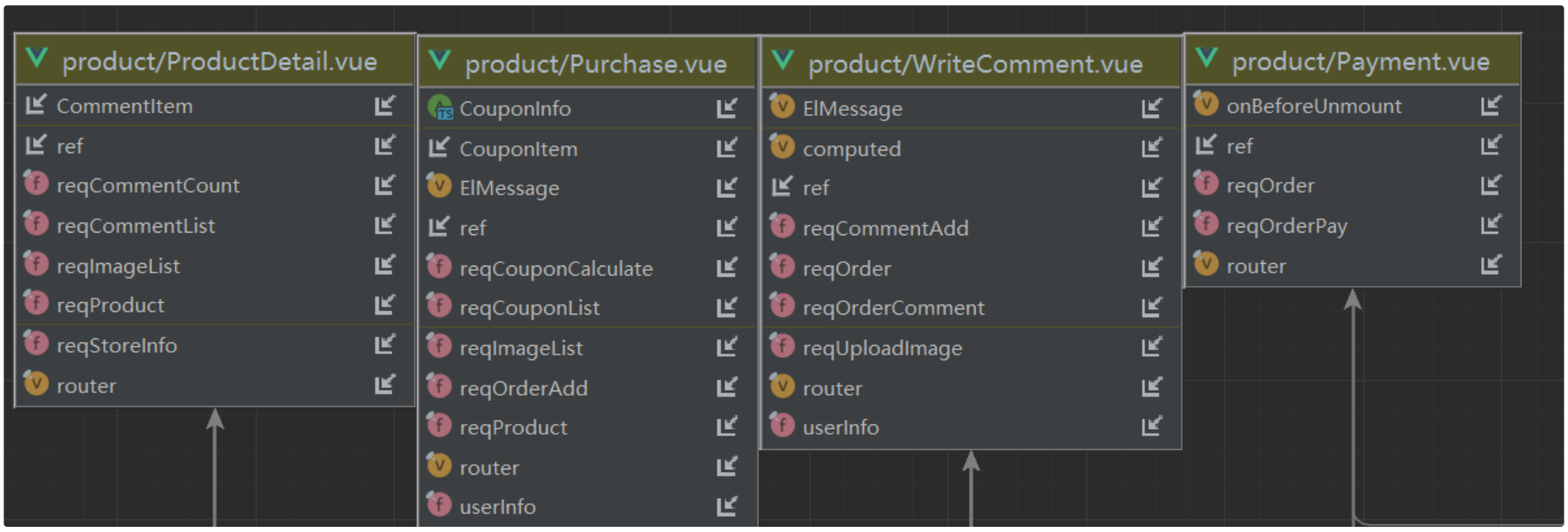
productUI包括了商品创建、访问，和商品信息高度绑定的页面，以及实现页面所需要的业务逻辑和网络请求。

4.1.3.2 整体结构

- 包括页面跳转图中的相关页面(淡紫色)
- productUI会调用相关网络API包向后端发送HTTP请求

视图类列表

名称	路径
商品详情界面	/src/views/product/ProductDetail.vue
订单创建界面	/src/views/product/Purchase.vue
支付界面	/src/views/product/Payment.vue
评论界面	/src/views/product/WriteComment.vue



4.1.3.3 模块内部类的接口规范

src/api/**product.ts**

名称	请求方法	接口参数	说明
reqProductList	GET	{storeId: number}	通过商店id请求商店列表
reqProduct	GET	{itemId: number}	通过商品id请求单个商品信息
reqProductCategory	GET		请求商品分类列表
reqProductAdd	POST	{id: number,name: string,description?: string,price: number,category: string,storeId: number,inventory: number}	请求创建商品
reqUpdateInventory	POST	{itemId:number, inventory:number}	请求更新库存
reqSearchProduct	POST	{page: number,name?: string,category?: [string],minPrice?: number,maxPrice?: number,priceOrder?: string}	请求搜索商品

src/api/**comment.ts**

名称	请求方法	接口参数	说明
reqCommentList	GET	{itemId: number, startInd: number}	请求评论列表
reqCommentCount	GET	{itemId: number}	请求评论总数
reqCommentAdd	POST	{id: number,content: string,itemId: number,userId: number,orderId: number,rating: number}	新增评论

4.1.4 othersUI

4.1.4.1 模块概述

一些功能比较零散的页面

4.1.4.2 整体结构

- 用户界面层包括页面跳转图中的相关页面(灰色)

视图类列表

名称	路径
404界面	/src/views/others/NotFound.vue
重定向界面	/src/views/others/Home.vue

4.1.4.3 模块内部类的接口规范

无

4.2 业务逻辑层的分解

业务逻辑层的开发包图

4.2.1 userbl

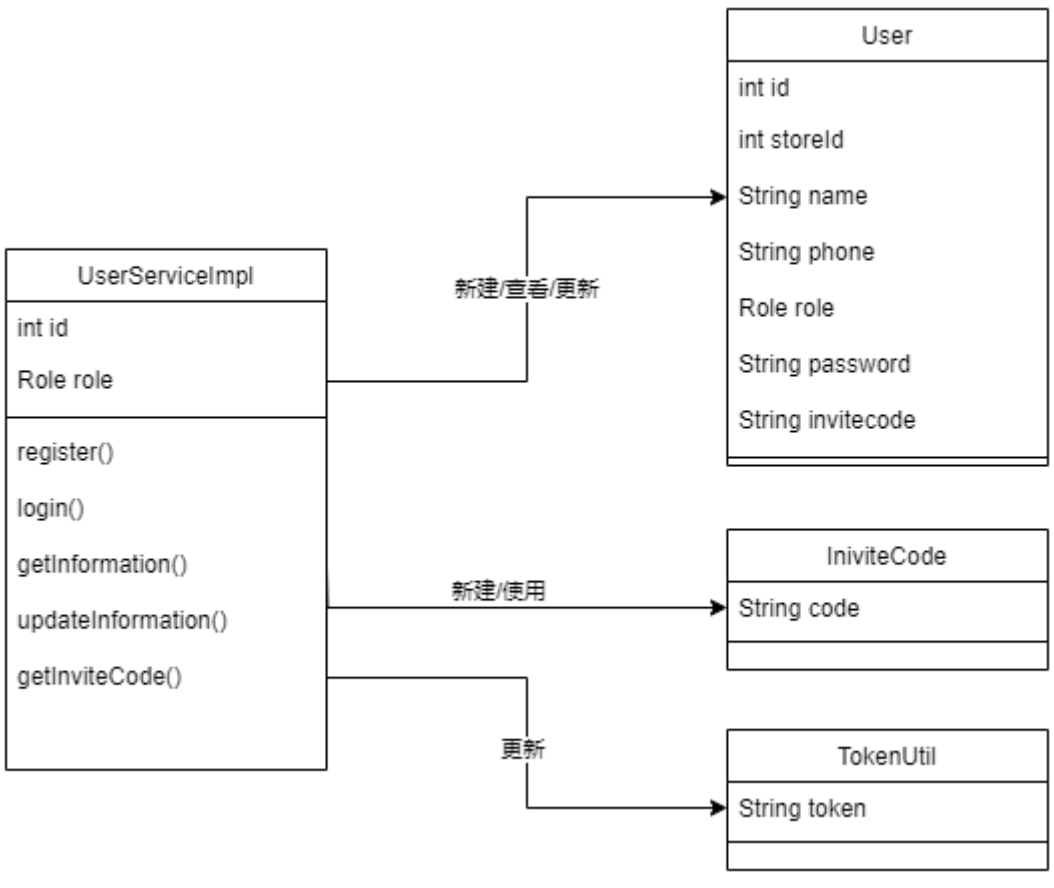
4.2.1.1 模块概述

userbl模块承担的需求参加需求规格说明文档功能需求及相关非功能需求。

userbl模块的职责及接口参见软件系统结构描述文档。

4.2.1.2 整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口，比如业务逻辑层和数据层之间我们添加UserRepository接口等。为了隔离业务逻辑职责和逻辑控制职责，我们增加了UserController，这样UserController会将销售的业务逻辑处理委托给User对象。UserPO是作为用户的持久化对象被添加到模型设计中去的。



模块	职责
UserService	负责用户界面需要的模块

4.2.1.3 模块内部类的接口规范

提供的服务（供接口）

名称	语法	前置条件	后置条件
UserService.register	public Boolean register(UserVO userVO)	注册信息符合输入规则	新建保存一个持久化对象User
UserService.login	public String login(String phone,String password)	电话和密码信息符合输入规则	查找是否存在对应的User，根据输入的密码返回登录令牌
UserService.getInformation	public UserVO getInformation();	无	调用数据库返回用户VO
UserService.updateInformation	public Boolean updateInformation(UserVO userVO);	用户更新信息符合规则	更新数据库中持久对象的内容
UserService.getInviteCode	public String getInviteCode(RoleEnum role);	role是合法角色	返回一个邀请码

需要的服务（需接口）

服务名	服务
UserRepository.findByPhone(String phone)	根据phone查找单一持久化对象
UserRepository.save(User user)	保存单一持久化对象
UserRepository.findByPhoneAndPassword(String phone,String password)	根据phone和password查找单一持久化对象
InviteRepository.findByCode(String code)	根据code查找单一持久对象
InviteRepository.delete(String code)	删除单一持久化对象
TokenUtil.getToken(User user)	获取用户令牌
SecurityUtil.getCurrentUser()	获取当前登录用户信息
InviteCodeRepository.save(InviteCode inviteCode)	保存单一持久化对象

4.2.1.4 业务逻辑层的设计原理

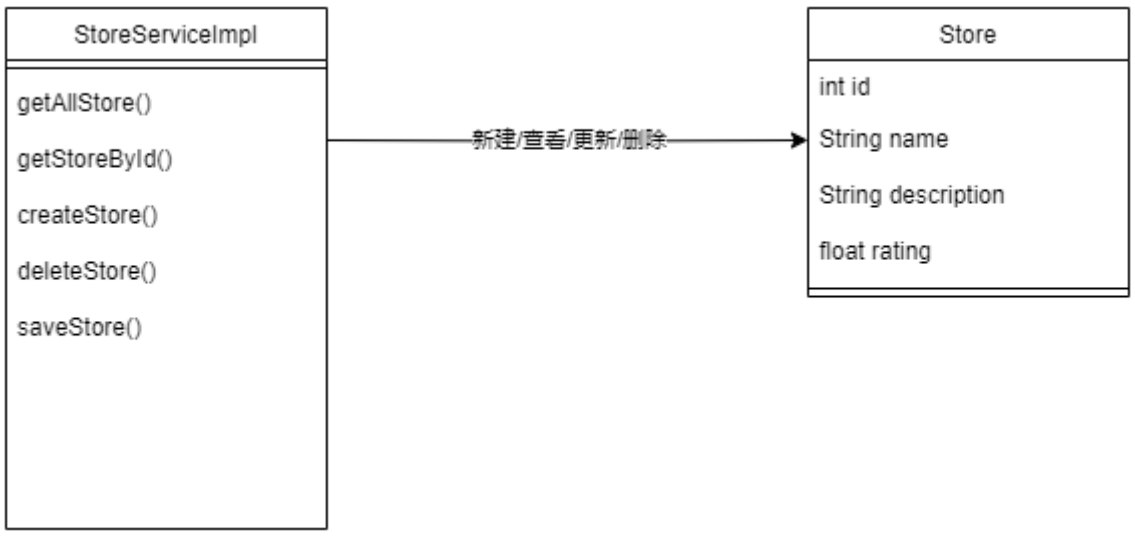
利用委托式控制风格，每个界面需要访问的业务逻辑由各自的控制器委托给不同领域的对象。

4.2.2 storebl

4.2.2.1 模块概述

storebl模块承担的需求参加需求规格说明文档功能需求及相关非功能需求。

storebl模块的职责及接口参见软件系统结构描述文档。



模块	职责
StoreService	负责商店界面需要的模块

4.2.2.2 整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口，比如业务逻辑层和数据层之间我们添加StoreRepository接口等。为了隔离业务逻辑职责和逻辑控制职责，我们增加了StoreController，这样StoreController会将销售的业务逻辑处理委托给Store对象。StorePO是作为用户的持久化对象被添加到模型设计中去的。

4.2.2.3 模块内部类的接口规范

提供的服务（供接口）

名称	语法	前置条件	后置条件
StoreService.getAllStore	public List getAllStore()	无	返回所有商店VO
StoreService.getStoreVOById	public StoreVO getStoreVOById(int storeId)	提供的商店id合法有效	返回对应商店VO
StoreService.createStore	public Integer createStore(StoreVO storeVO)	输入商店信息符合规则	创建一个持久化商店对象
StoreService.deleteStore	public boolean deleteStore(int storeId)	提供的商店id合法有效	删除对应的持久化商店对象
StoreService.saveStore	public void saveStore(Store store}		
	StorePO正常	保存单一持久化对象	

需要的服务（需接口）

服务名	服务
StoreRepository.findAll()	查找Store表中所有持久化对象
StoreRepository.findById(int storeId)	根据商店id查找单一持久化对象
StoreRepository.findByName(String name)	根据商店名称查找单一持久化对象
StoreRepository.save(Store store)	保存单一持久化对象
StoreRepository.delete(Store store)	删除单一持久化对象

4.2.2.4 业务逻辑层的设计原理

利用委托式控制风格，每个界面需要访问的业务逻辑由各自的控制器委托给不同领域的对象。

4.2.3 itembl

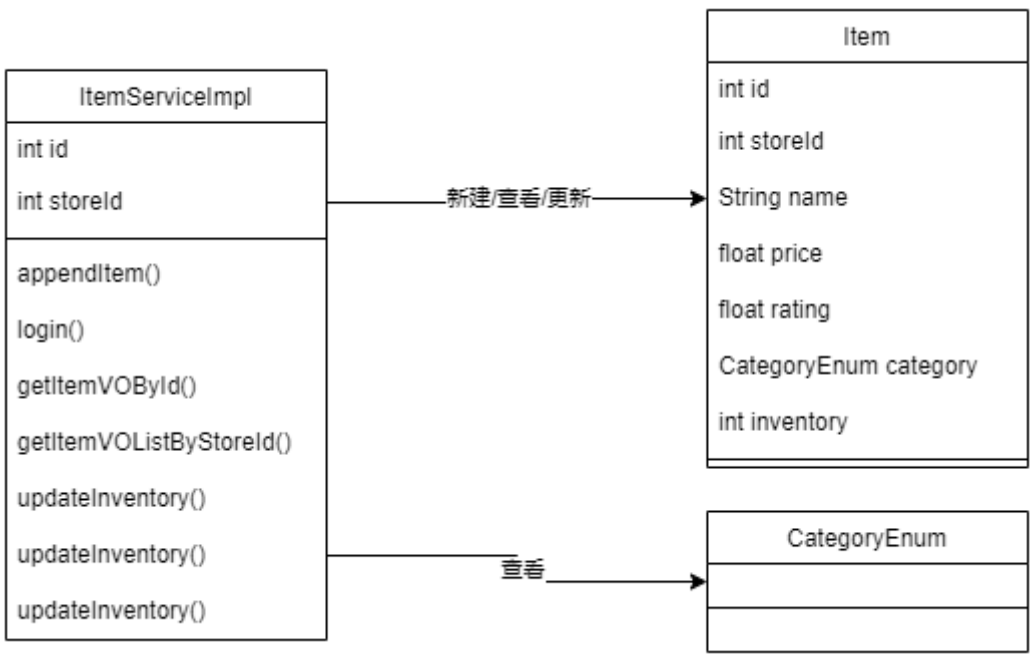
4.2.3.1 模块概述

itembl模块承担的需求参加需求规格说明文档功能需求及相关非功能需求。

itembl模块的职责及接口参见软件系统结构描述文档。

4.2.3.2 整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口，比如业务逻辑层和数据层之间我们添加ItemRepository接口等。为了隔离业务逻辑职责和逻辑控制职责，我们增加了ItemController，这样ItemController会将销售的业务逻辑处理委托给Item对象。ItemPO是作为用户的持久化对象被添加到模型设计中去的。



模块	职责
ItemService	负责商品界面需要的模块

4.2.3.3 模块内部类的接口规范

提供的服务（供接口）

名称	语法	前置条件	后置条件
ItemService.appendItem	public Integer appendItem(ItemVO itemVO)	商品信息符合规则	创建持久化商品对象并返回商品id
ItemService.getItemVOByItemId	public ItemVO getItemVOById(Integer itemId)	输入的商品id合法有效	返回对应商品VO
ItemService.getItemVOListByStoreId	public List getItemVOListByStoreId(Integer storeId)	输入的商店id合法有效	返回该商店所有商品VO
ItemService.updateInventory	public Boolean updateInventory(Integer itemId, Integer inventory)	输入的商品id与库存数量合法有效	更新持久化对象信息
ItemService.getItemCategories	public List getItemCategories()	无	返回商品分类的枚举类信息
ItemService.getItemVOByConditions	public Page getItemVOByConditions(Map<String, Object> params)	无	以PageList形式返回ItemVO
ItemService.saveItem	public void saveItem(Item item)	商品PO信息正常	保存单一持久化对象

需要的服务（需接口）

服务名	服务
ItemRepository.save(item)	保存单一持久化对象
ItemRepository.findById(int id)	根据商品id查找单一持久化对象
ItemRepository.findByStoreId(int storeId)	根据商店id查找多个持久化对象
SecurityUtil.getCurrentUser()	获取当前用户信息

4.2.3.4 业务逻辑层的设计原理

利用委托式控制风格，每个界面需要访问的业务逻辑由各自的控制器委托给不同领域的对象。

4.2.4 orderbl

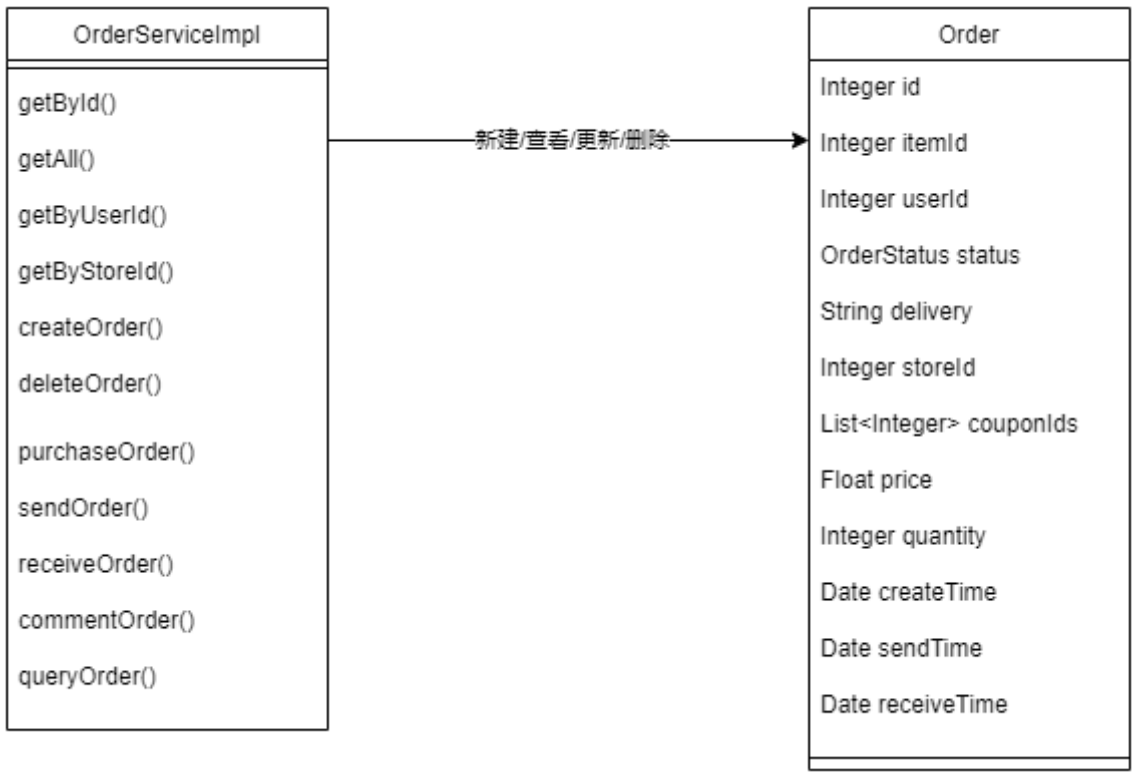
4.2.4.1 模块概述

orderbl模块承担的需求参加需求规格说明文档功能需求及相关非功能需求。

orderbl模块的职责及接口参见软件系统结构描述文档。

4.2.4.2 整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口，比如业务逻辑层和数据层之间我们添加OrderRepository接口等。为了隔离业务逻辑职责和逻辑控制职责，我们增加了OrderController，这样OrderController会将销售的业务逻辑处理委托给Order对象。OrderPO是作为用户的持久化对象被添加到模型设计中去的。



模块	职责
OrderService	负责订单界面需要的模块

4.2.4.3 模块内部类的接口规范

提供的服务（供接口）

名称	语法	前置条件	后置条件
OrderService.getById	public OrderVO getById(int orderId)	输入的订单id合法有效	返回对应订单的VO
OrderService.getAll	public List getAll()	当前用户为CEO或MANAGER	返回所有订单的VO
OrderService.getByUserId	public List getByUserId()	用户已登录	返回当前用户所有订单VO
OrderService..getByStoreId	public List getByStoreId()	用户有所属商店	返回当前用户所属商店的所有订单VO
OrderService.createOrder	public Integer createOrder(OrderVO orderVO)	操作用户为订单所有者且商品库存不为0	创建一个单一订单持久化对象并返回订单id
OrderService.deleteOrder	public Boolean deleteOrder(int orderId)	操作用户为订单所有者且操作订单有效	删除单一订单持久化对象
OrderService.purchaseOrder	public Boolean purchaseOrder(int orderId)	输入的订单id合法有效且订单状态为未付款	更新单一订单持久化对象状态为未发货
OrderService.sendOrder	public Boolean sendOrder(int orderId)	输入的订单id合法有效、操作订单归属于当前用户商店且状态为未发货	更新单一订单持久化对象状态为已发货
OrderService.receiveOrder	public Boolean receiveOrder(int orderId)	输入的订单id合法有效、操作用户为订单所有者且订单状态为未收货	更新单一订单持久化对象状态为已收货
OrderService.commentOrder	public Boolean commentOrder(int orderId)	输入的订单id合法有效、操作用户为订单所有者且订单状态为未评论	更新单一订单持久化对象状态为已评论
OrderService.queryOrder	public List queryOrder(int storeId)	当前用户为CEO或门店工作人员	返回查询的商店所有订单VO

需要的服务（需接口）

服务名	服务
OrderRepository.findById(int id)	根据订单id查找单一持久化对象
OrderRepository.findAll()	查找所有订单持久化对象
OrderRepository.findByUserId(int userId)	根据用户id查找多个持久化对象
OrderRepository.findByStoreId(int storeId)	根据商店id查找多个持久化对象
OrderRepository.save(Order order)	保存单一持久化对象
OrderRepository.delete(Order order)	删除单一持久化对象
SecurityUtil.getCurrentUser()	获取当前用户信息
SecurityUtil.getCurrentUserRole()	获取当前用户角色身份
SecurityUtil.getCurrentUserId()	获取当前用户id
ItemService.getItemById (int id)	根据商品id查找单一持久化对象
CouponService.clacPrice(int couponId,float price)	根据优惠券及原始价格计算使用优惠券后的价格

4.2.4.4 业务逻辑层的设计原理

利用委托式控制风格，每个界面需要访问的业务逻辑由各自的控制器委托给不同领域的对象。

4.2.5 commentbl

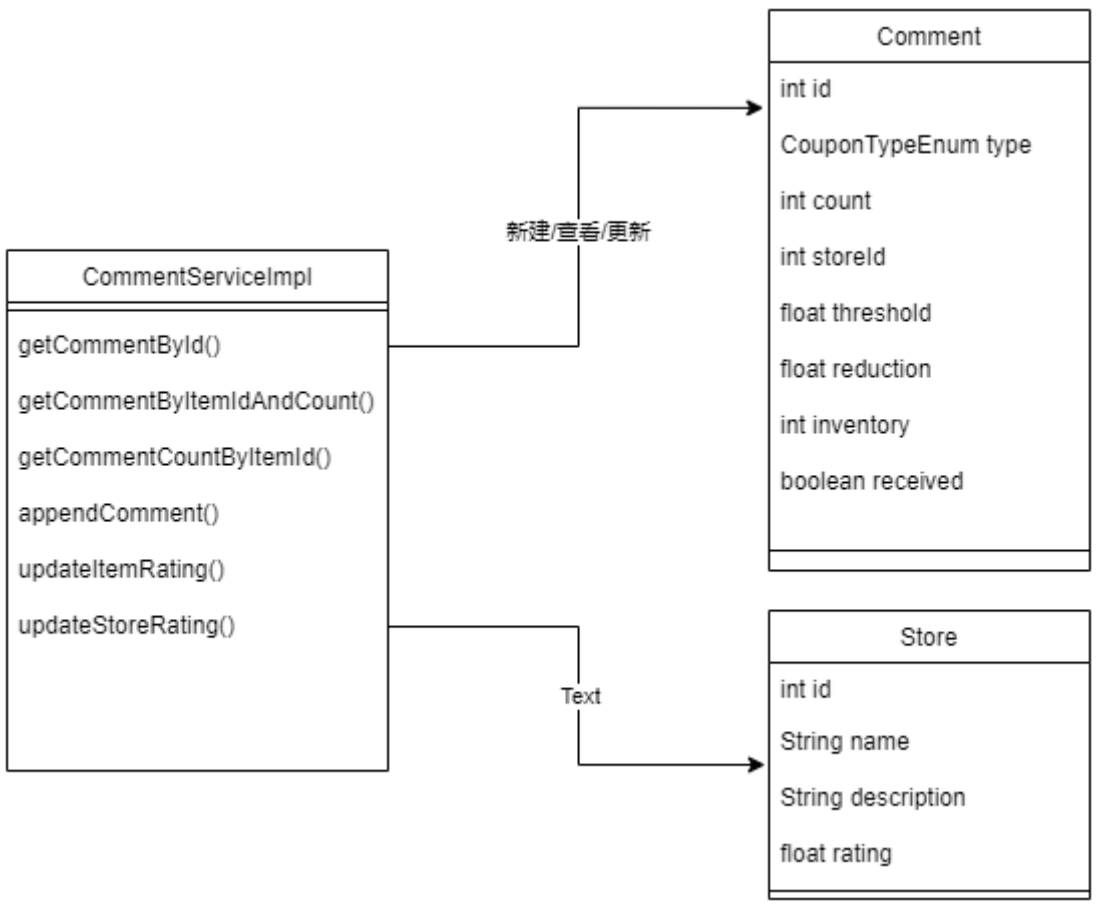
4.2.5.1 模块概述

commentbl模块承担的需求参加需求规格说明文档功能需求及相关非功能需求。

commentbl模块的职责及接口参见软件系统结构描述文档。

4.2.5.2 整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口，比如业务逻辑层和数据层之间我们添加CommentRepository接口等。为了隔离业务逻辑职责和逻辑控制职责，我们增加了CommentController，这样CommentController会将销售的业务逻辑处理委托给Comment对象。CommentPO是作为用户的持久化对象被添加到模型设计中去的。



模块	职责
CommentService	负责评论界面需要的模块

4.2.5.3 模块内部类的接口规范

提供的服务（供接口）

名称	语法	前置条件	后置条件
CommentService.getCommentById	public CommentVO getCommentById(Integer commentId)	输入的id是合法有效的	返回对应的评论VO
CommentService.getCommentByItemAndCount	public List getCommentByItemIdAndCount(Integer itemId, Integer startInd)	输入的商品id和index是合法有效的	返回对应商品对应数量的评论VO
CommentService.getCommentByItemId	public Integer getCommentCountByItemId(Integer itemId)	输入的商品id是合法有效的	返回对应商品的所有评论VO
CommentService.appendComment	public Integer appendComment(CommentVO commentVO)	输入的评论信息是合法有效的	创建持久化对象,持久化更新对应商品及商店的评分,并返回该评论的id
CommentService.updateItemRating	private void updateItemRating(Integer itemId, float rating)	输入的商品id以及新增评价的评分是合法有效的	持久化更新该商品评分
CommentService.updateStoreRating	private void updateStoreRating(Integer itemId, float rating)	输入的商店id以及新增评价的评分是合法有效的	持久化更新该商店评分

需要的服务（需接口）

服务名	服务
CommentRepository.findById(int id)	根据评论id查找单一持久化对象
CommentRepository.findByItemId(int ItemId)	根据商品id查找多个持久化对象
CommentRepository.save(Comment comment)	保存单一持久化对象
SecurityUtil.getCurrentUser()	查询当前用户信息
ItemService.getItemById(int itemId)	根据商品id查找单一持久化对象
ItemService.getItemListByStoreId(int storeId)	根据商店id查找多个持久化对象
ItemService.saveItem(Item item)	保存单一持久化对象
StoreService.getStoreVOById(int storeId)	根据商店id查找对应商店VO
StoreService.saveStore(Store store)	保存单一持久化对象

4.2.5.4 业务逻辑层的设计原理

利用委托式控制风格，每个界面需要访问的业务逻辑由各自的控制器委托给不同领域的对象。

4.2.6 Couponbl

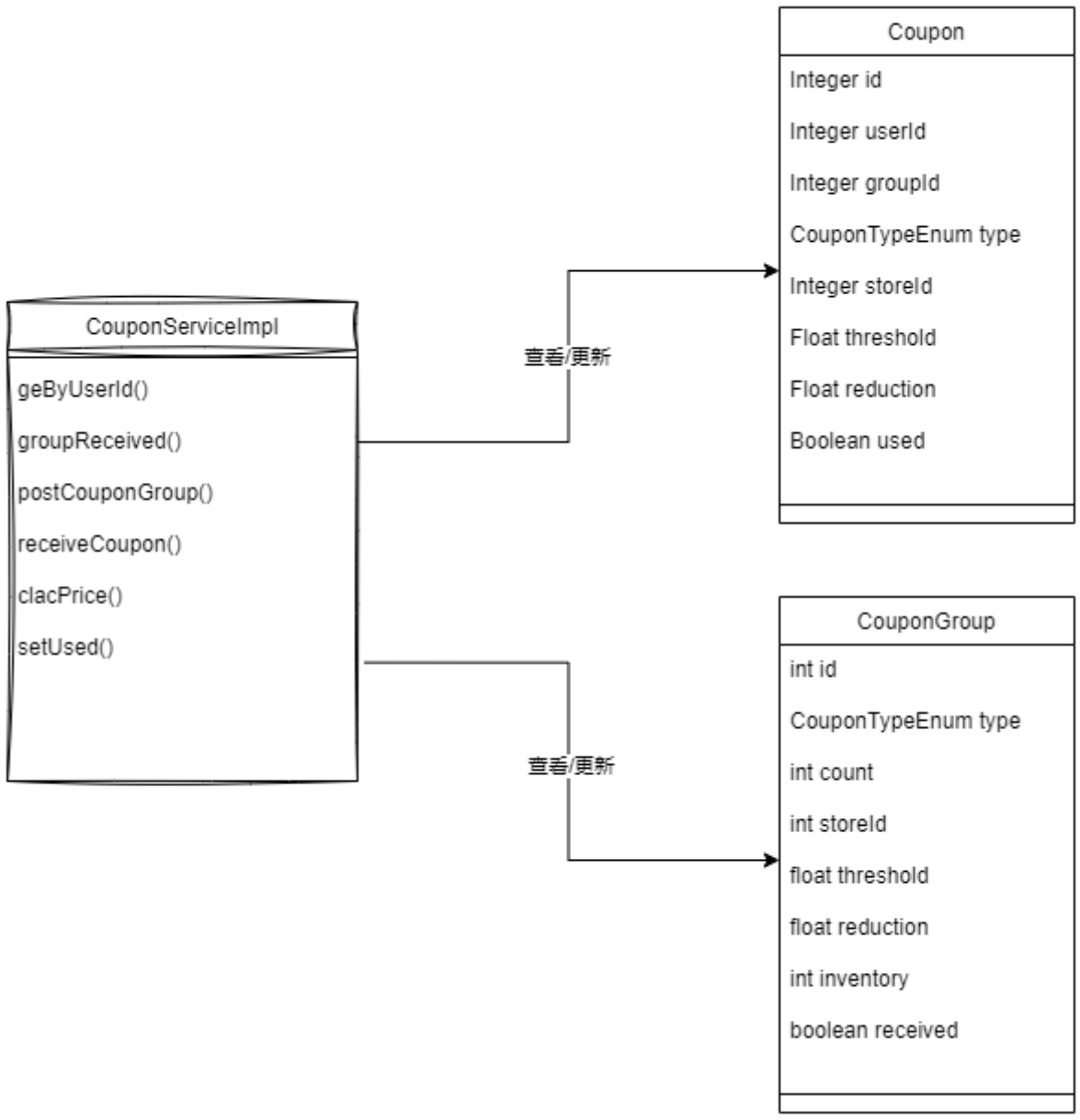
4.2.6.1 模块概述

couponbl模块承担的需求参加需求规格说明文档功能需求及相关非功能需求。

couponbl模块的职责及接口参见软件系统结构描述文档。

4.2.6.2 整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口，比如业务逻辑层和数据层之间我们添加CouponRepository接口等。为了隔离业务逻辑职责和逻辑控制职责，我们增加了CouponController，这样CouponController会将销售的业务逻辑处理委托给Coupon对象。CouponPO是作为用户的持久化对象被添加到模型设计中去的。



模块	职责
CouponService	负责优惠券界面需要的模块

4.2.6.3 模块内部类的接口规范

提供的服务（供接口）

名称	语法	前置条件	后置条件
CouponService.getByUserId	public List getByUserId()	用户已登陆	返回当前用户发表的所有评论VO
CouponService.groupReceived	public boolean groupReceived(int groupId)	用户已登录及优惠券组id合法有效	返回当前用户是否领取过该优惠券
CouponService.receiveCoupon	public boolean receiveCoupon(int groupId)	用户已登录、该优惠券组id对应的优惠券组存在且库存不为0	保存优惠券单一持久化对象，更新优惠券组持久化对象，并返回”领取成功“
CouponService. clacPrice	public float clacPrice(int couponId, float price)	该优惠券id对应优惠券存在	返回使用优惠券后的价格
CouponService.setUsed	public void setUsed(int couponId)	该优惠券id对应优惠券存在	更新单一优惠券持久化对象

需要的服务（需接口）

服务名	服务
SecurityUtil.getCurrentUser().getId()	获取当前用户id
CouponRepository.findByUserId(int userId)	根据用户id查找该用户所有已领取的优惠券
CouponRepository.findFirstByUserIdAndGroupId(int userId,int groupId)	查询该用户最先领取的优惠券组为GroupId的优惠券
CouponGroupService.getCouponGroupById(int groupId)	根据groupId查找对应的优惠券
CouponRepository.save(coupon)	保存/更新对应的单一持久化对象
couponGroupService.useCouponGroup(int groupId)	保存/更新对应的单一持久化对象
CouponContext.calculate(coupon, price)	计算使用优惠券后的价格

4.2.6.4 业务逻辑层的设计原理

利用委托式控制风格，每个界面需要访问的业务逻辑由各自的控制器委托给不同领域的对象。

4.2.7 CouponGroupbl

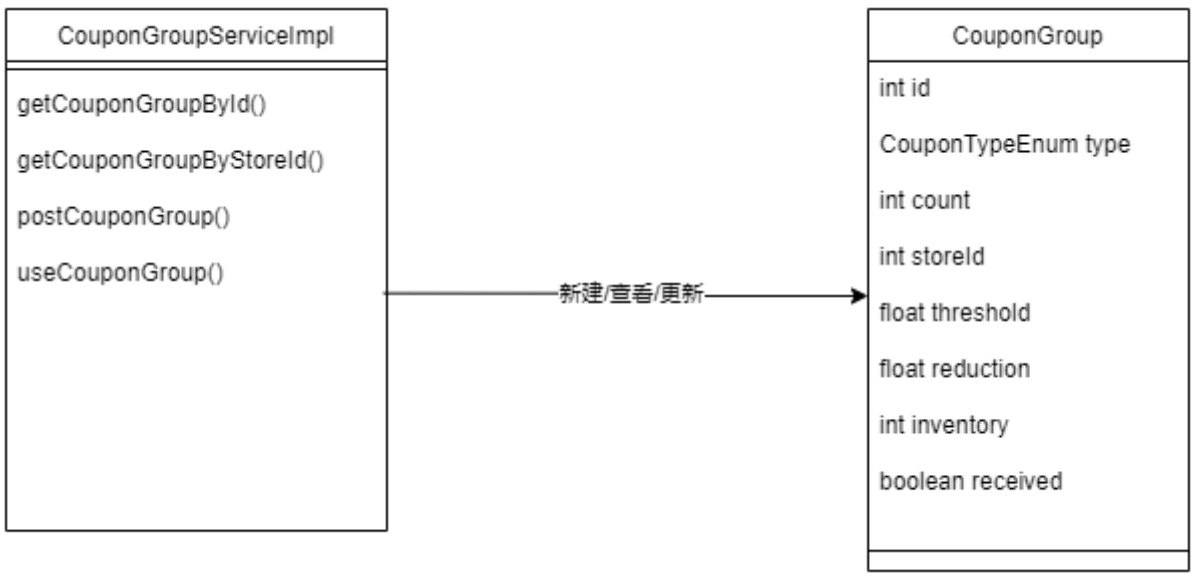
4.2.7.1 模块概述

CouponGroupbl模块承担的需求参加需求规格说明文档功能需求及相关非功能需求。

CouponGroupbl模块的职责及接口参见软件系统结构描述文档。

4.2.7.2 整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口，比如业务逻辑层和数据层之间我们添加CouponGroupRepository接口等。为了隔离业务逻辑职责和逻辑控制职责，我们增加了CouponGroupController，这样CouponGroupController会将销售的业务逻辑处理委托给CouponGroup对象。CouponGroupPO是作为用户的持久化对象被添加到模型设计中去的。



模块	职责
CouponGroupService	负责优惠券组界面需要的模块

4.2.7.3 模块内部类的接口规范

提供的服务（供接口）

名称	语法	前置条件	后置条件
CouponGroup.getCouponGroupByStoreId	public List getCouponGroupByStoreId(int storeId)	输入的商店（全局）id合法且有效	返回对应商店（全局）所有优惠券组
CouponGroupService.postCouponGroup	public int postCouponGroup(CouponGroupVO couponGroupVO)	用户为STAFF或CEO且优惠券组信息符合规则	创建一个单一优惠券组持久化对象并返回其id
CouponGroupService.useCouponGroup	public void useCouponGroup(int groupId)	该优惠券组库存不为0	更新一个单一优惠券组持久化对象

需要的服务（需接口）

服务名	服务
SecurityUtil.getCurrentUser()	获取当前用户信息
CouponGroupRepository.findAll()	获取所有优惠券组
CouponGroupRepository.findAllByStoreId(int storeId)	根据商店id查找多个持久化对象
CouponServiceImpl.groupReceived(int id)	获取用户是否领取该优惠券组信息
CouponGroupRepository.save(CouponGroup CG)	保存单一持久化对象

4.2.7.4 业务逻辑层的设计原理

利用委托式控制风格，每个界面需要访问的业务逻辑由各自的控制器委托给不同领域的对象。

4.2.8 reportbl

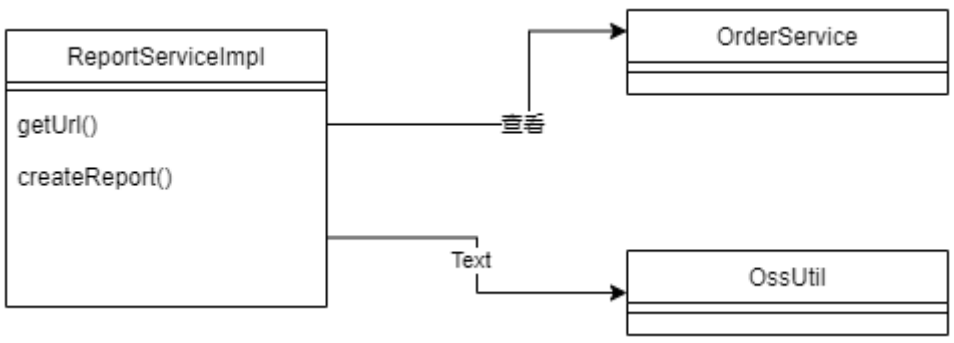
4.2.8.1 模块概述

reportbl模块承担的需求参加需求规格说明文档功能需求及相关非功能需求。

reportbl模块的职责及接口参见软件系统结构描述文档。

4.2.8.2 整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口，比如业务逻辑层和数据层之间我们添加ReportRepository接口等。为了隔离业务逻辑职责和逻辑控制职责，我们增加了ReportController，这样ReportController会将销售的业务逻辑处理委托给Report对象。ReportPO是作为用户的持久化对象被添加到模型设计中去的。



模块	职责
ReportService	负责报表界面需要的模块

4.2.8.3 模块内部类的接口规范

提供的服务（供接口）

名称	语法	前置条件	后置条件
ReportServ.getUrl	public String getUrl(int storeId) }	当前用户身份为CEO/STAFF	以Excel返回对应的订单报表

需要的服务（需接口）

服务名	服务
OrderService.queryOrder(int storeId)	根据商店id获取该商店所有的订单信息
OssUtil.upload(String name,InputStream is)	上传文件

4.2.8.4 业务逻辑层的设计原理

利用委托式控制风格，每个界面需要访问的业务逻辑由各自的控制器委托给不同领域的对象。

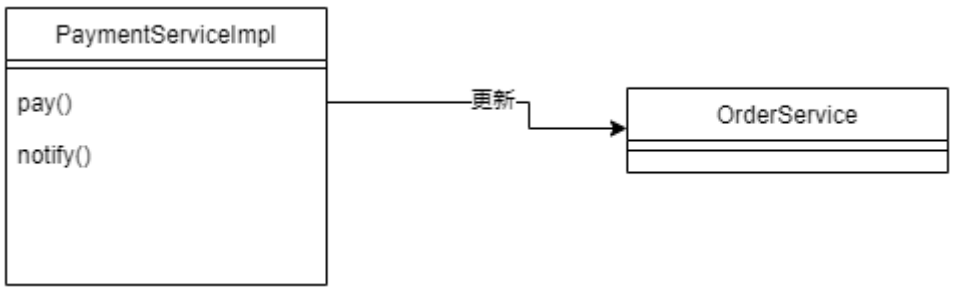
4.2.9 paymentbl

4.2.9.1 模块概述

paymentbll模块承担的需求参加需求规格说明文档功能需求及相关非功能需求。

paymentbl模块的职责及接口参见软件系统结构描述文档。

4.2.9.2 整体结构



模块	职责
PaymentService	负责支付界面需要的模块

4.2.9.3 模块内部类的接口规范

提供的服务（供接口）

名称	语法	前置条件	后置条件
PaymentService.pay	public void pay(AliPay aliPay, HttpServletResponse httpResponse) throws Exception		
PaymentService.notify	public String notify(HttpServletRequest request) throws Exception		
PaymentService.returnUrl	public String returnUrl()		

需要的服务（需接口）

服务名	服务
OrderService.purchaseOrder(orderId)	将对应的订单状态改为已支付

4.2.9.4 业务逻辑层的设计原理

利用委托式控制风格，每个界面需要访问的业务逻辑由各自的控制器委托给不同领域的对象。

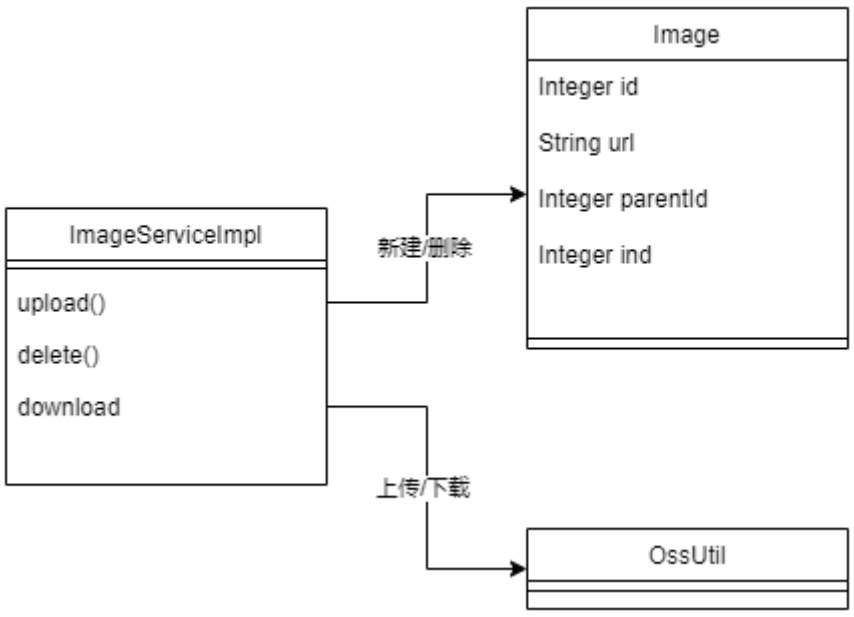
4.2.10 imagebl

4.2.10.1 模块概述

imagebl模块承担的需求参加需求规格说明文档功能需求及相关非功能需求。

imagebl模块的职责及接口参见软件系统结构描述文档。

4.2.10.2 整体结构



模块	职责
ImageService	负责图片界面需要的模块

4.2.10.3 模块内部类的接口规范

提供的服务（供接口）

ImageService.upload	public String upload(MultipartFile file, ImageType type, int parentId, int ind)	输入的图片信息有效	将图片上传至oss并且更新image数据库
ImageService.delete	public Boolean delete(ImageType type, int parentId)	要删除的图片信息存在	同时删除oss和数据库中的图片信息
ImageService.download	public List download(ImageType type, int parentId)	要下载的图片信息存在	读取数据库中图片的url并返回

需要的服务（需接口）

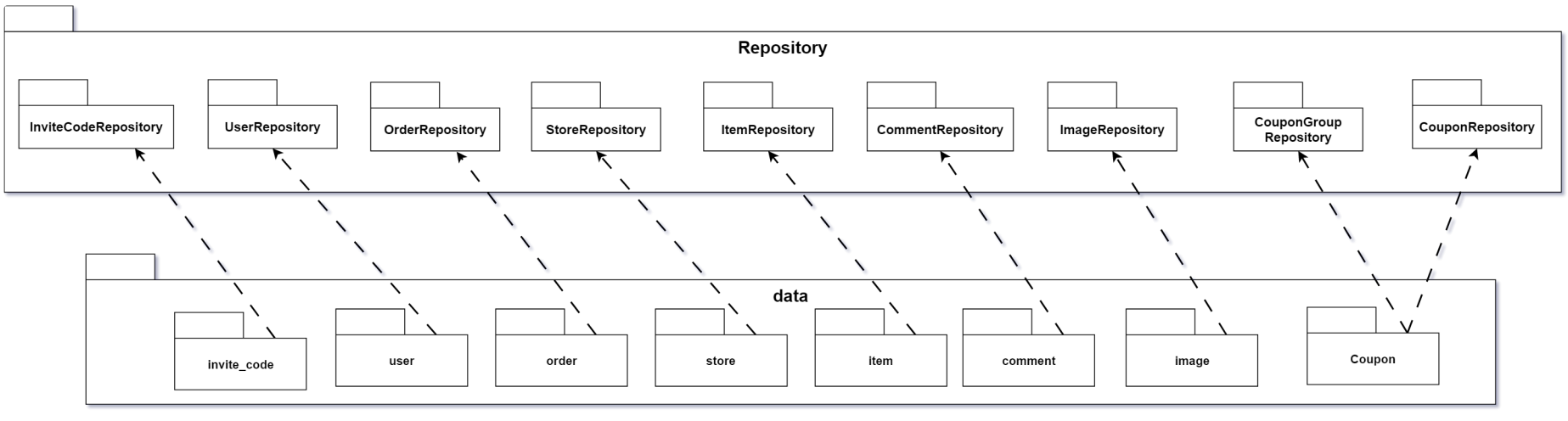
服务名	服务
OssUtil.upload(String objectName,InputStream inputStream)	上传图片到oss
OssUtil.delete(String objectUrl)	根据url删除oss中的图片
ImageRepository.findAllByTypeAndParentId(ImageType type,int parentId)	根据图片类型和父辈id查找多个持久化对象
ImageRepository.save(Image image)	保存单个持久化对象
ImageRepository.delete(Image image)	删除单个持久化对象

4.2.10.4 业务逻辑层的设计原理

利用委托式控制风格，每个界面需要访问的业务逻辑由各自的控制器委托给不同领域的对象。

4.3 数据访问层的分解

数据访问层对应后端Repository层与数据库



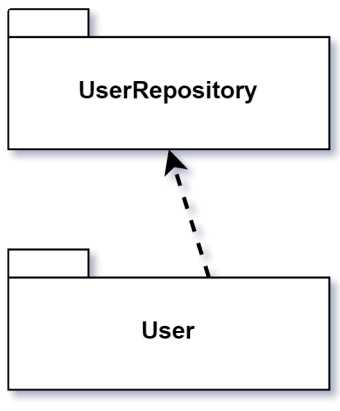
下面以Userdata为例

4.3.1 Userdata

4.3.1.1 模块概述

对应UserRepository以及User数据库。Repository是访问数据库的接口，由JPA实现。

4.3.1.2 整体结构



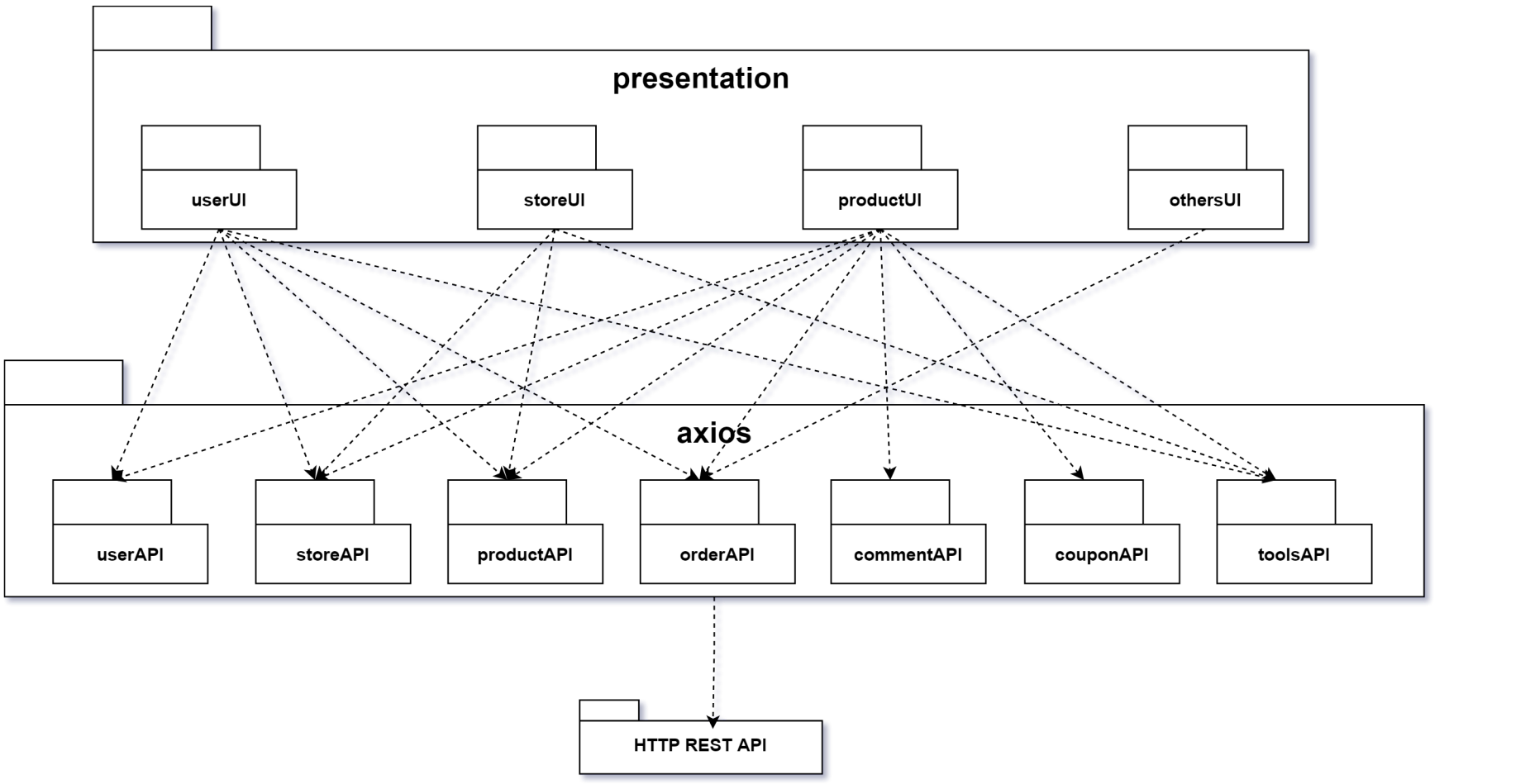
4.3.1.3 模块内部类的接口规范

接口规范除了JPA原有的接口之外，还有如下：

名称	语法	前置条件	后置条件
UserRepository.findByPhone	User findByPhone(String phone)	数据库中存在phone相同的User	根据phone进行查找返回对应的UserPO
UserRepository.findByPhoneAndPassword	User findByPhoneAndPassword(String phone, String password)	数据库中存在phone和password相同的User	根据phone和password进行查找返回对应的UserPO

5. 依赖视角

客户端的开发包图



服务端的开发包图