# 秒杀系统说明文档

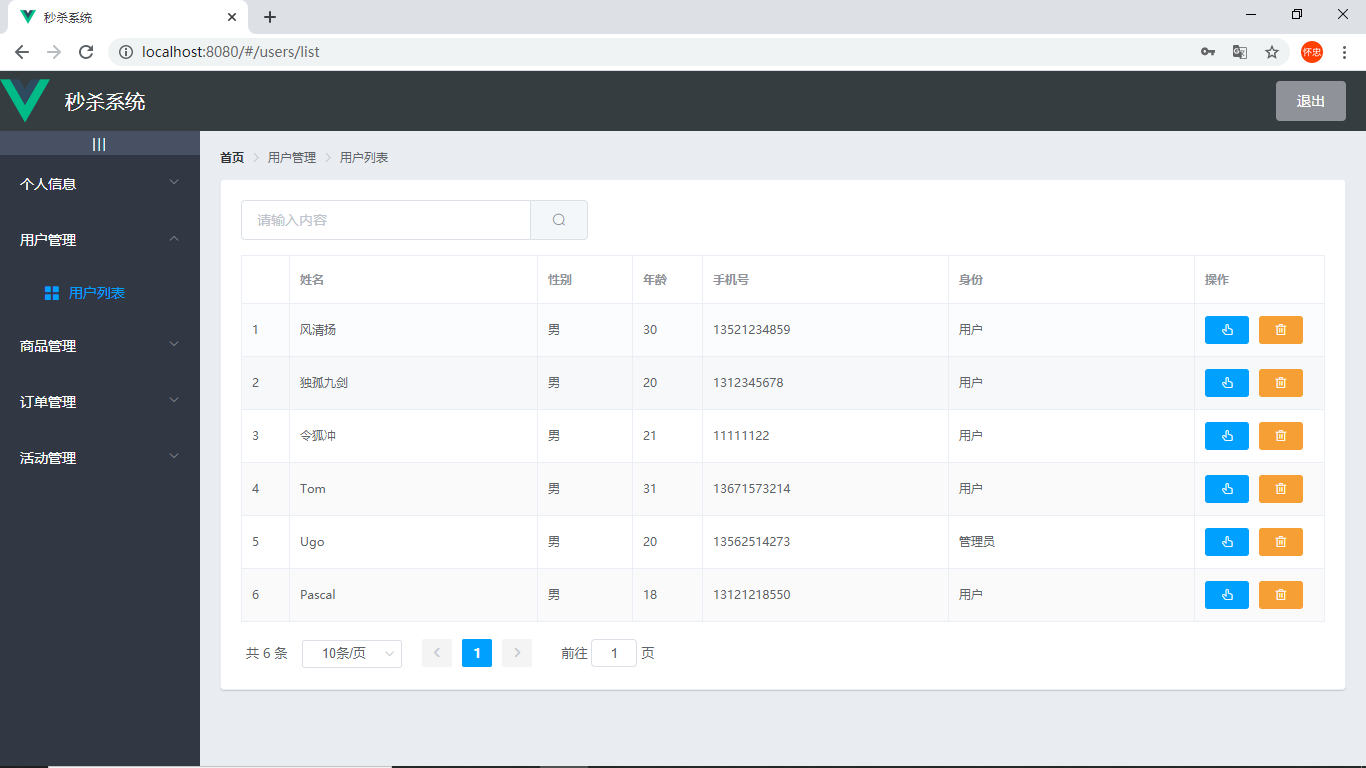
## 功能介绍

### 后台管理端

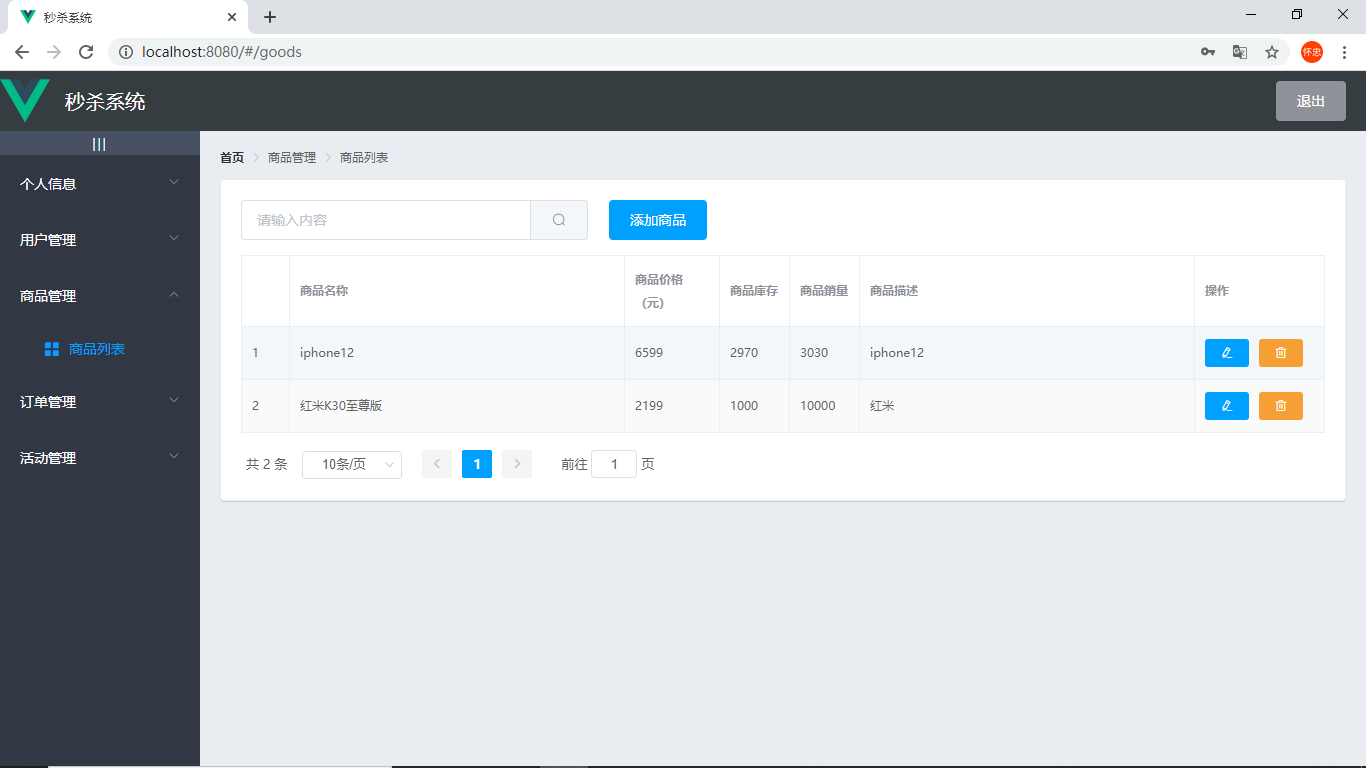
1、个人信息页



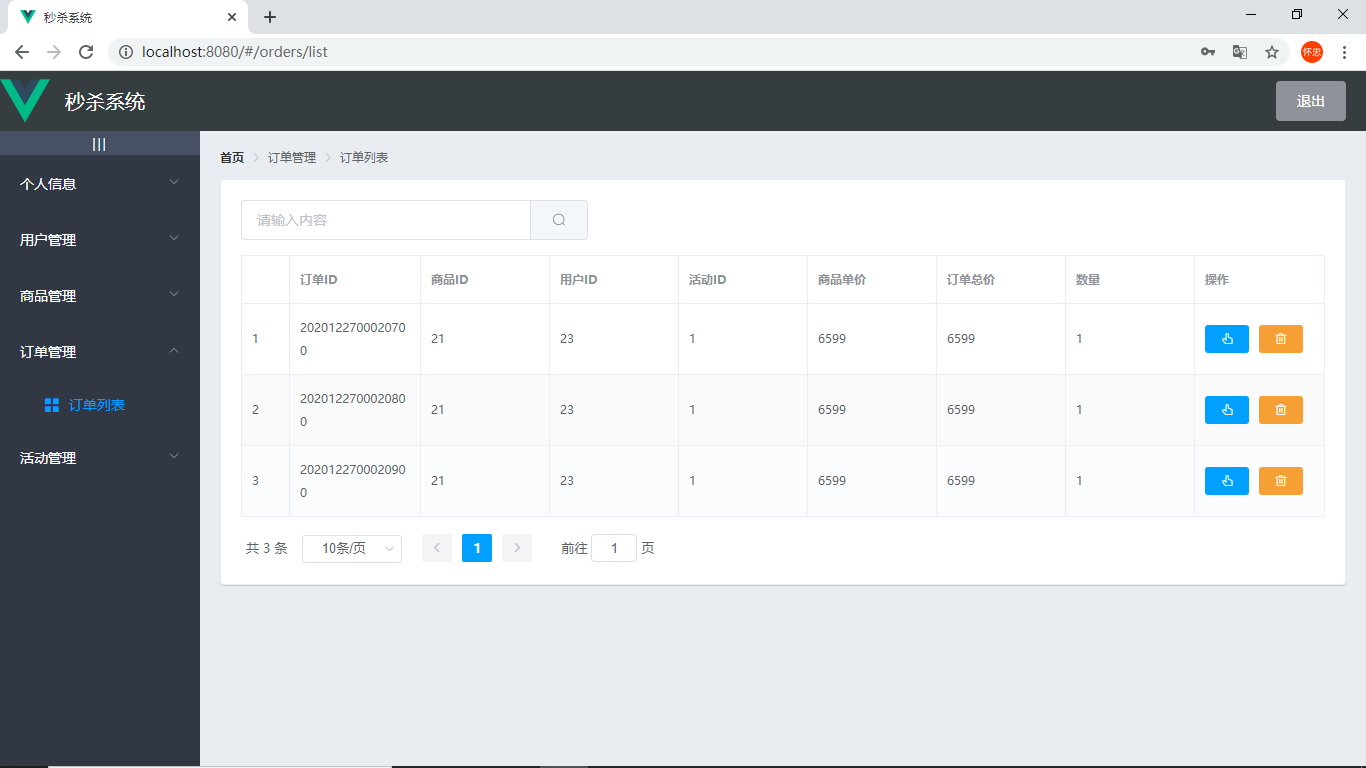
2、用户列表页



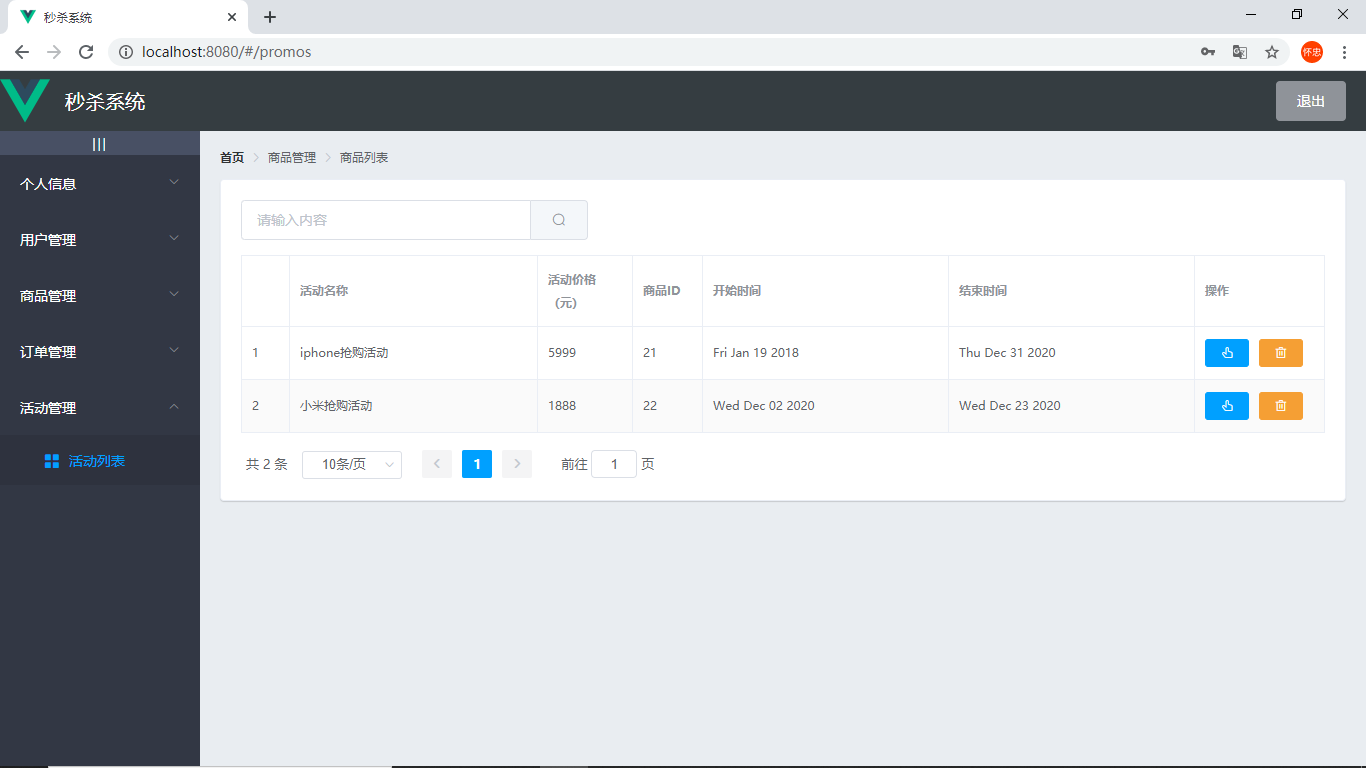
3、商品列表页



4、订单列表页

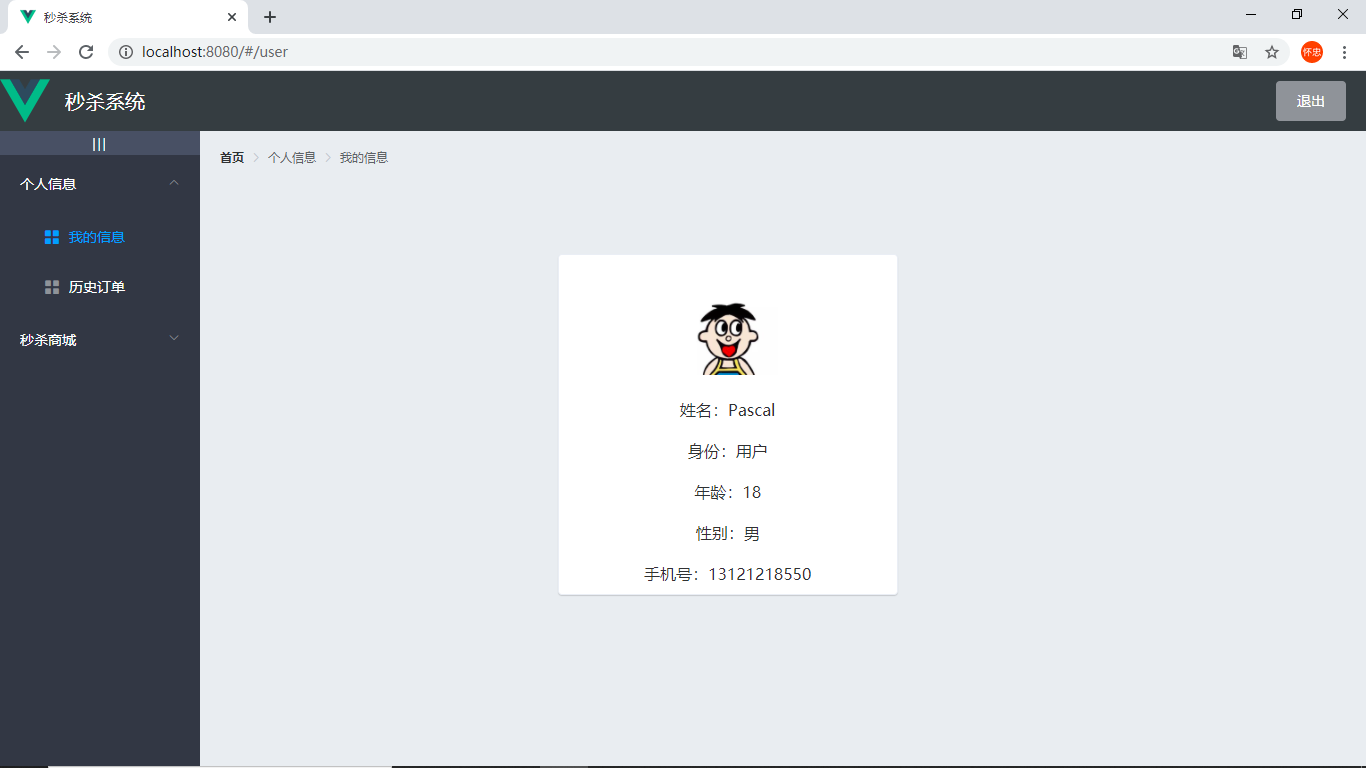


5、活动列表页

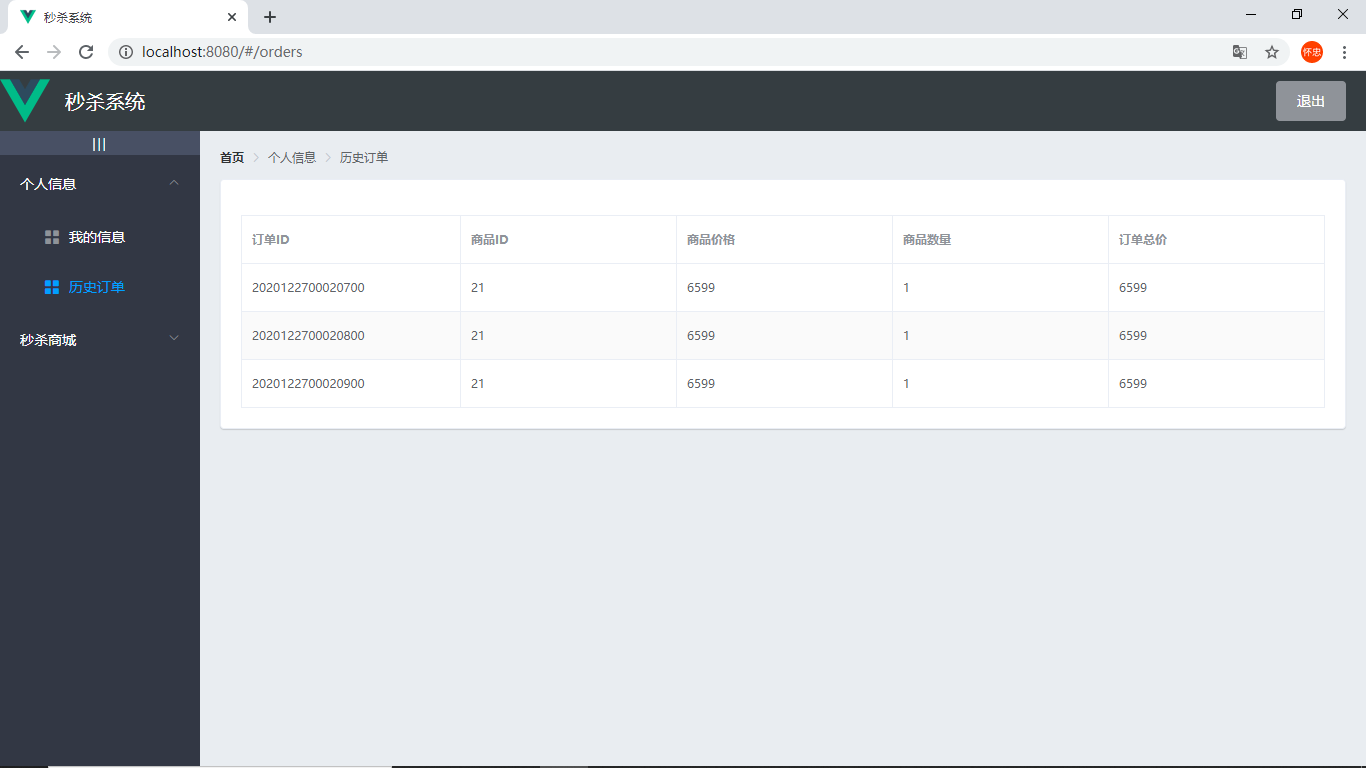


### 用户端

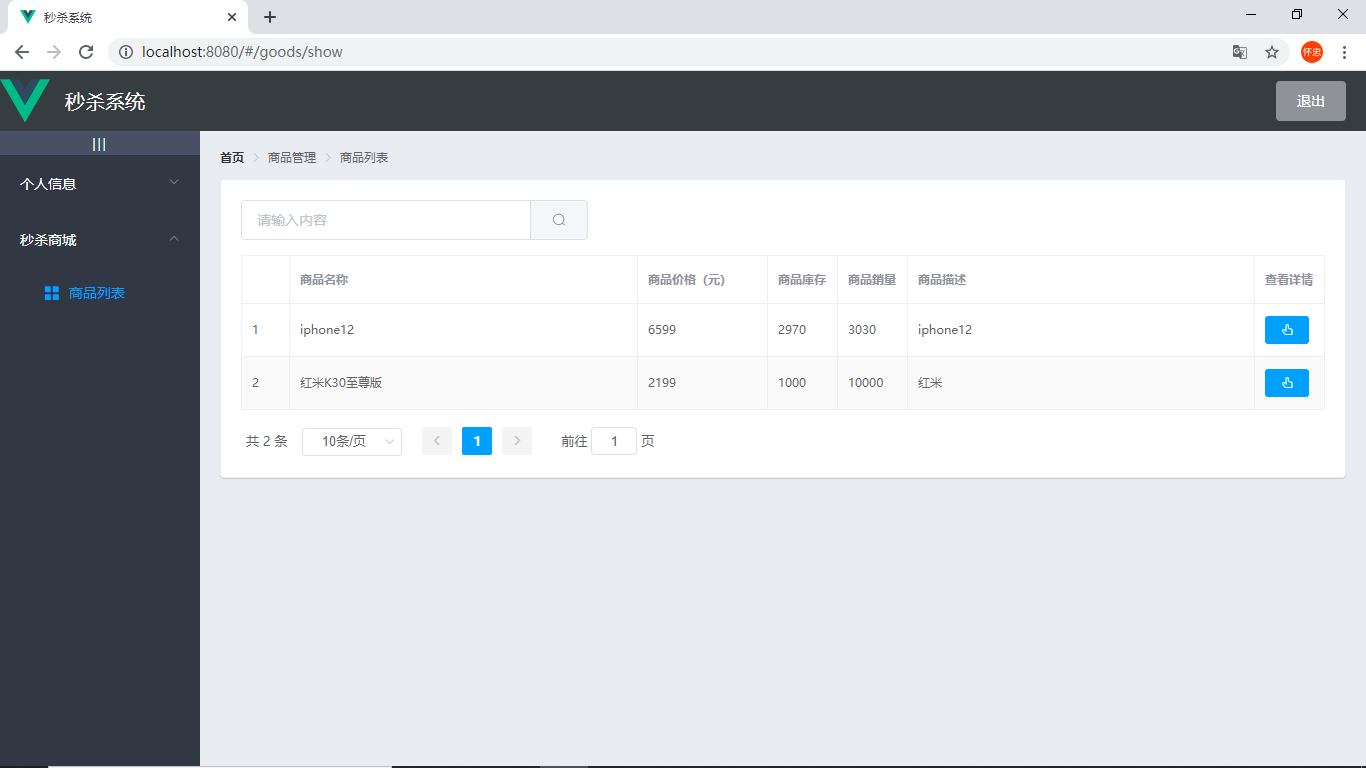
1、个人信息页



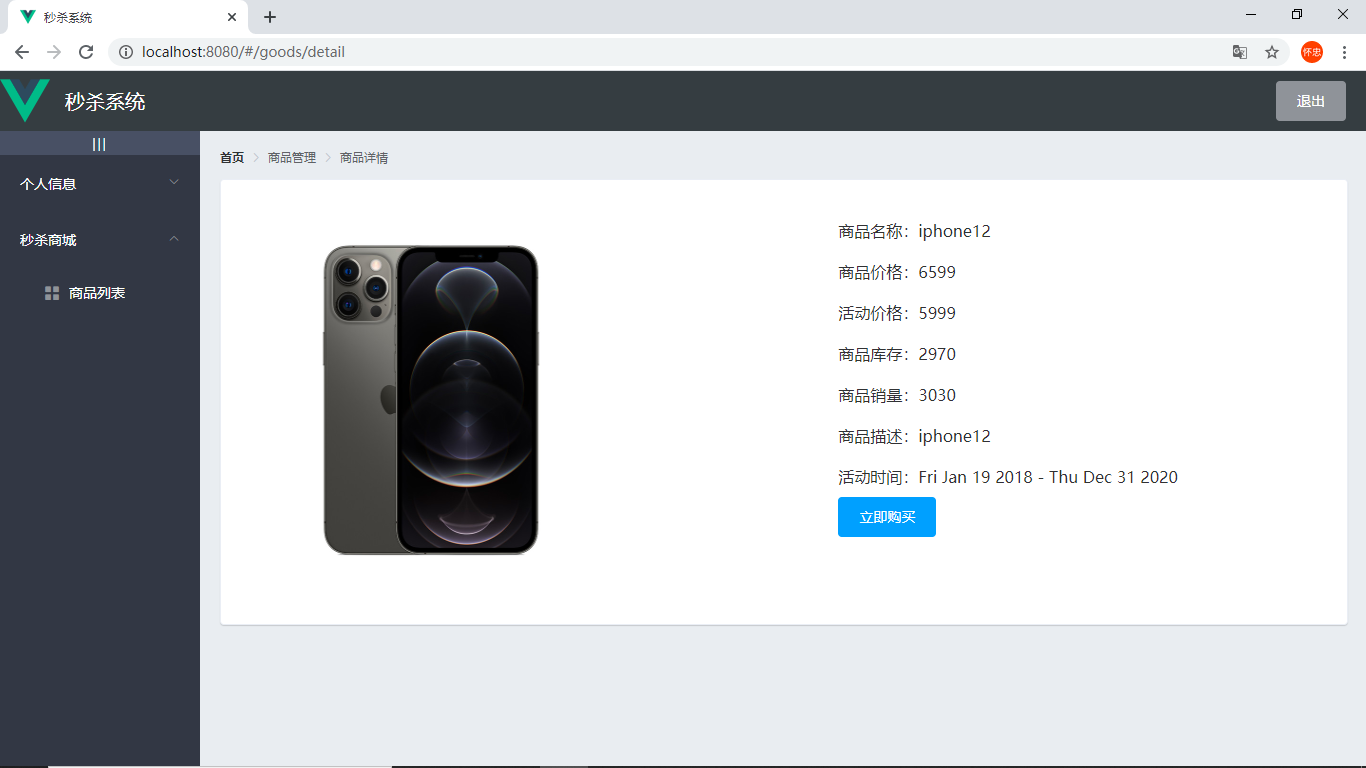
2、历史订单页



3、商品列表页



4、商品详情页

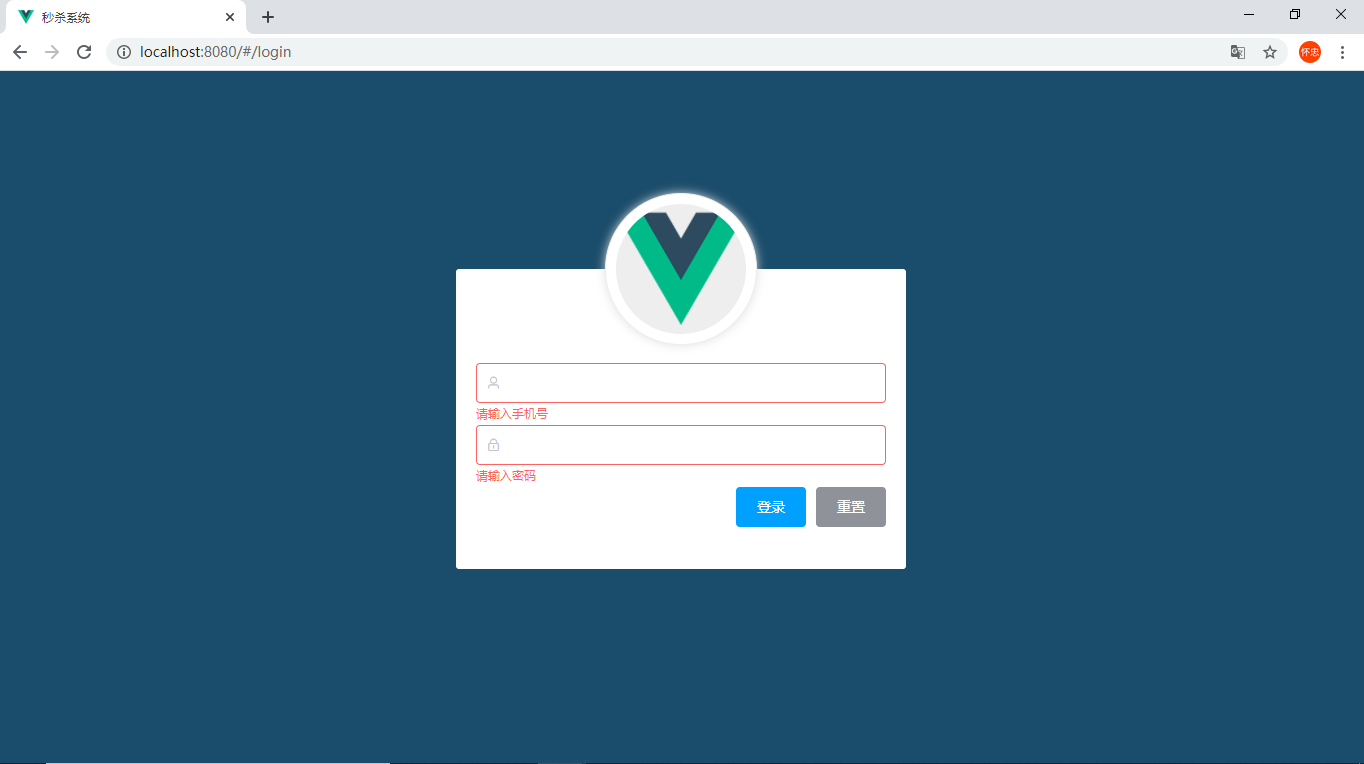


## 技术架构

本次课程设计的秒杀系统，前端使用Vue框架进行搭建，并采用Element UI组件进行开发。后端使用Java语言编写，采用SpringBoot搭建MVC框架。数据库采用MySQL进行数据存储，使用MyBatis管理数据库，并使用Alibaba Druid维护数据库连接池。后端使用双节点进行提供分布式服务，使用Nginx确保负载均衡。缓存采用Redis进行性能优化，通过RocketMQ消息队列维持缓存与数据库的一致性。

## 系统功能实现

#### 登录模块的设计与实现

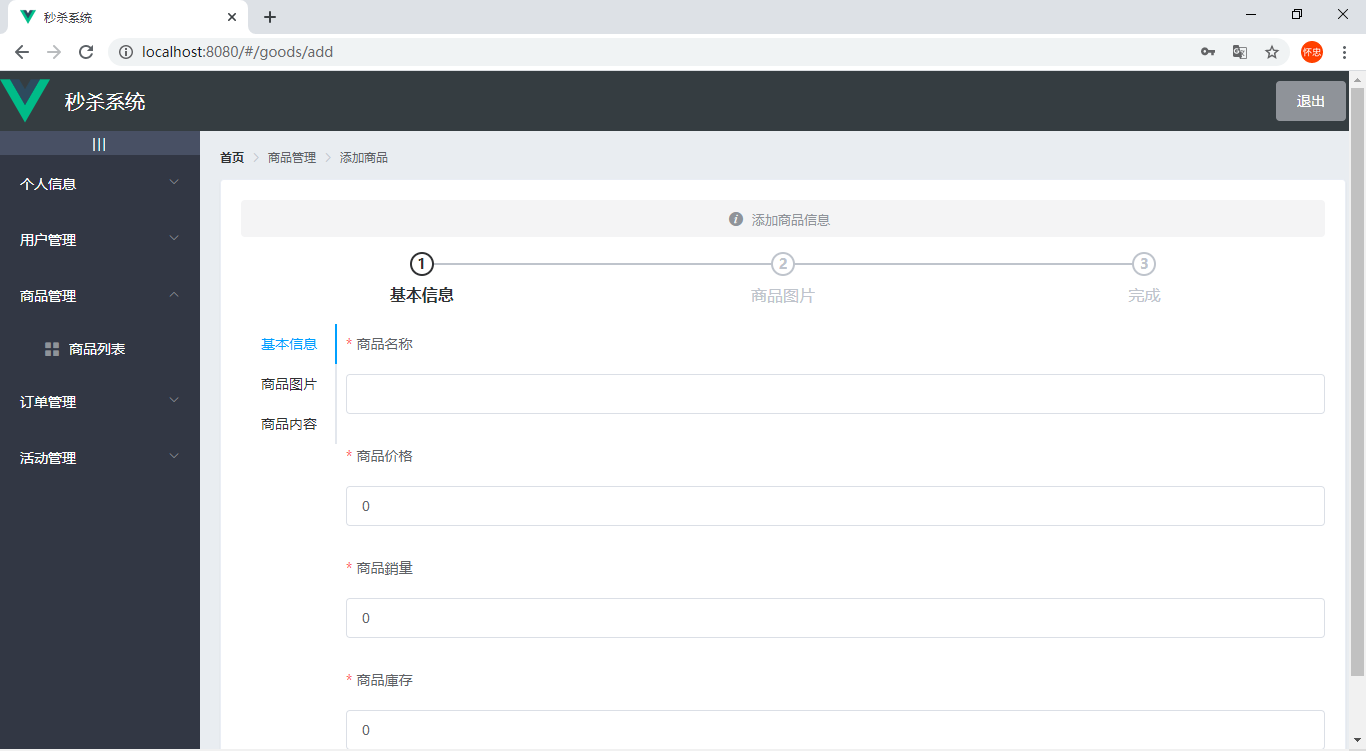


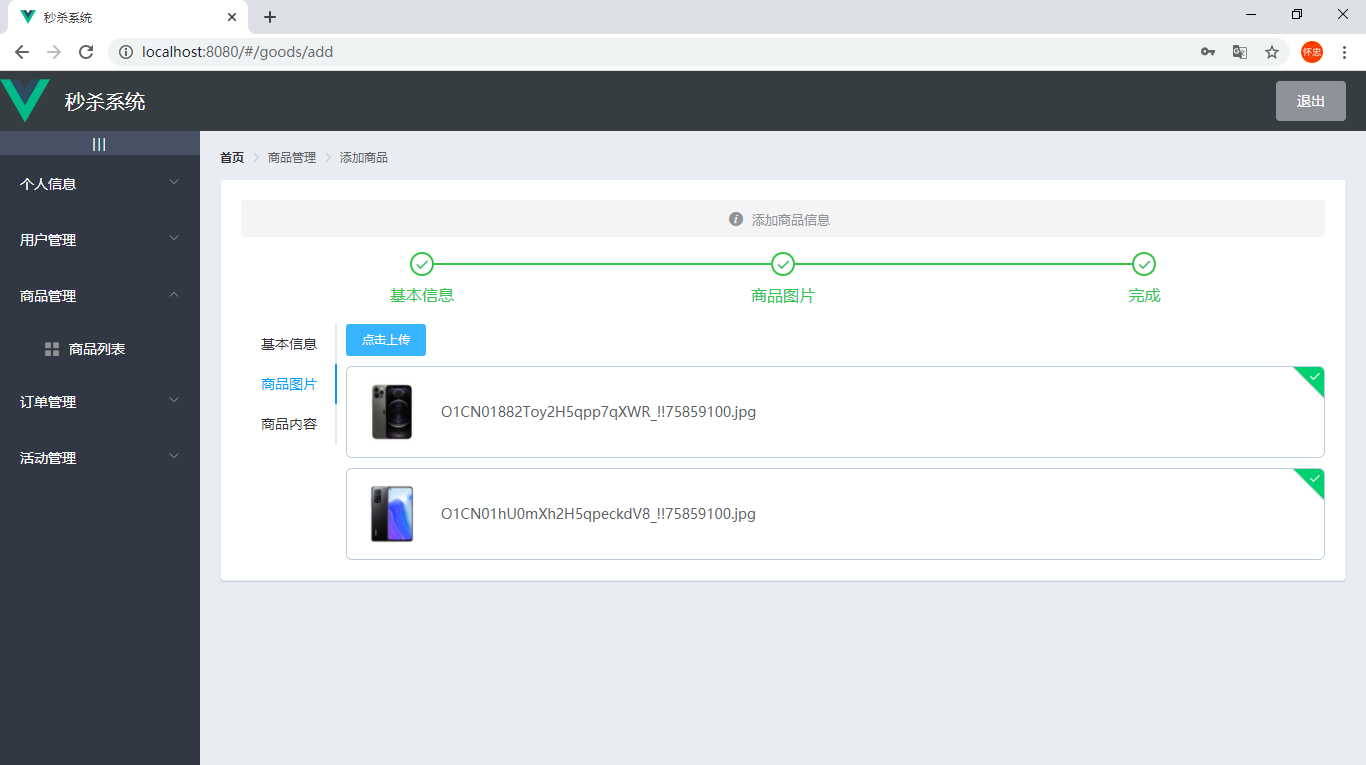
前端采用手机号与密码登录，并对输入格式进行校验。首先用户端向/user/token发起获取token请求，后端端根据手机号取出userModel，校验密码正确后，生成一个32位的随机数作为token，并作为键值与userModel一起存入redis，最后将token返回给用户端。用户端将token存入window.sessionStorage，在访问任何页面时，都必须将其作为凭证。

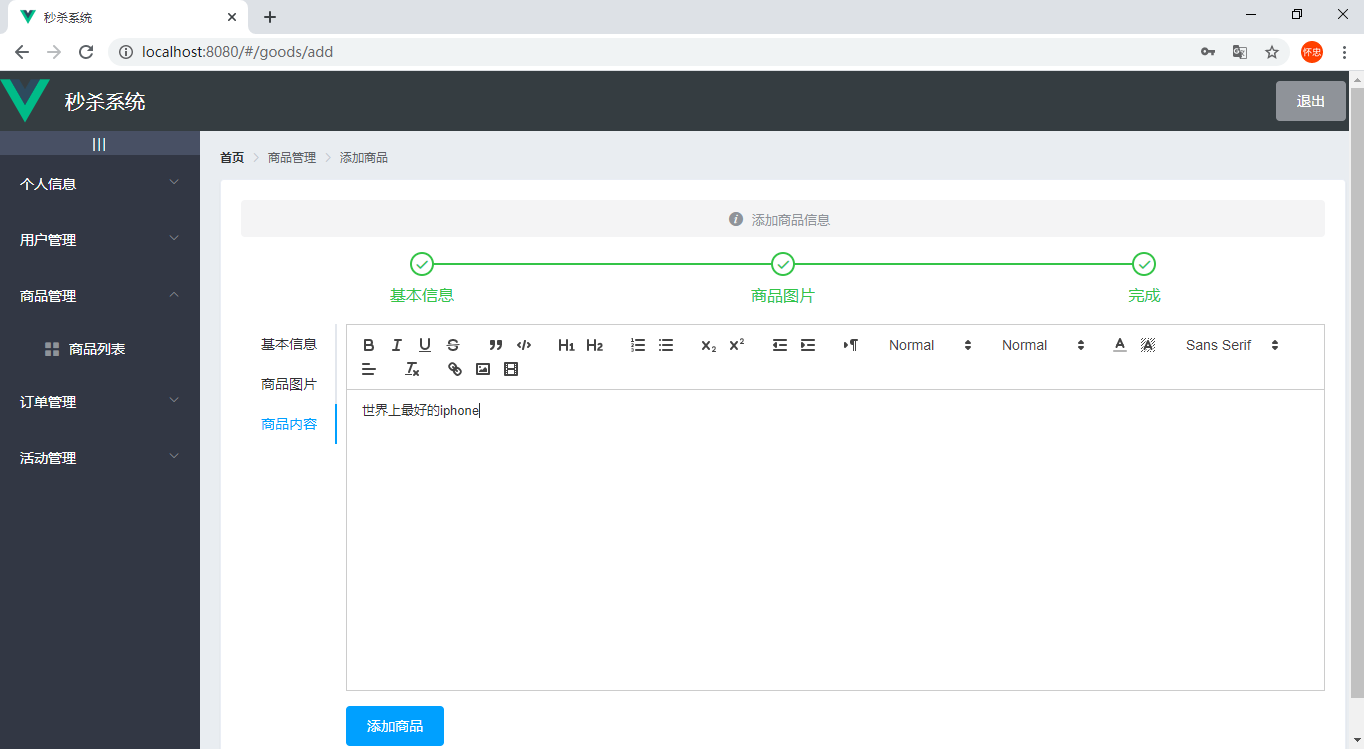
#### 后台管理页面增删改查模块的设计与实现

下面以商品管理页面为例进行介绍：

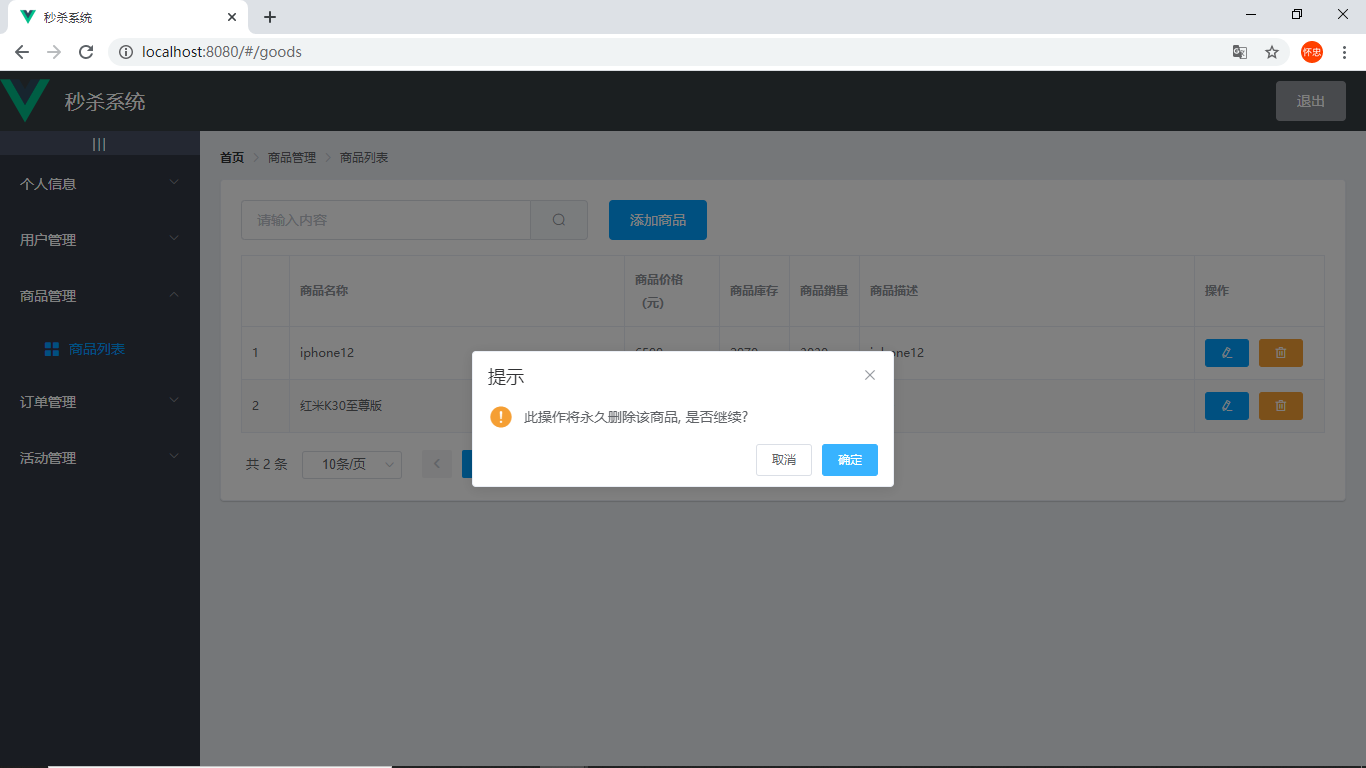
1、商品添加页面（增）



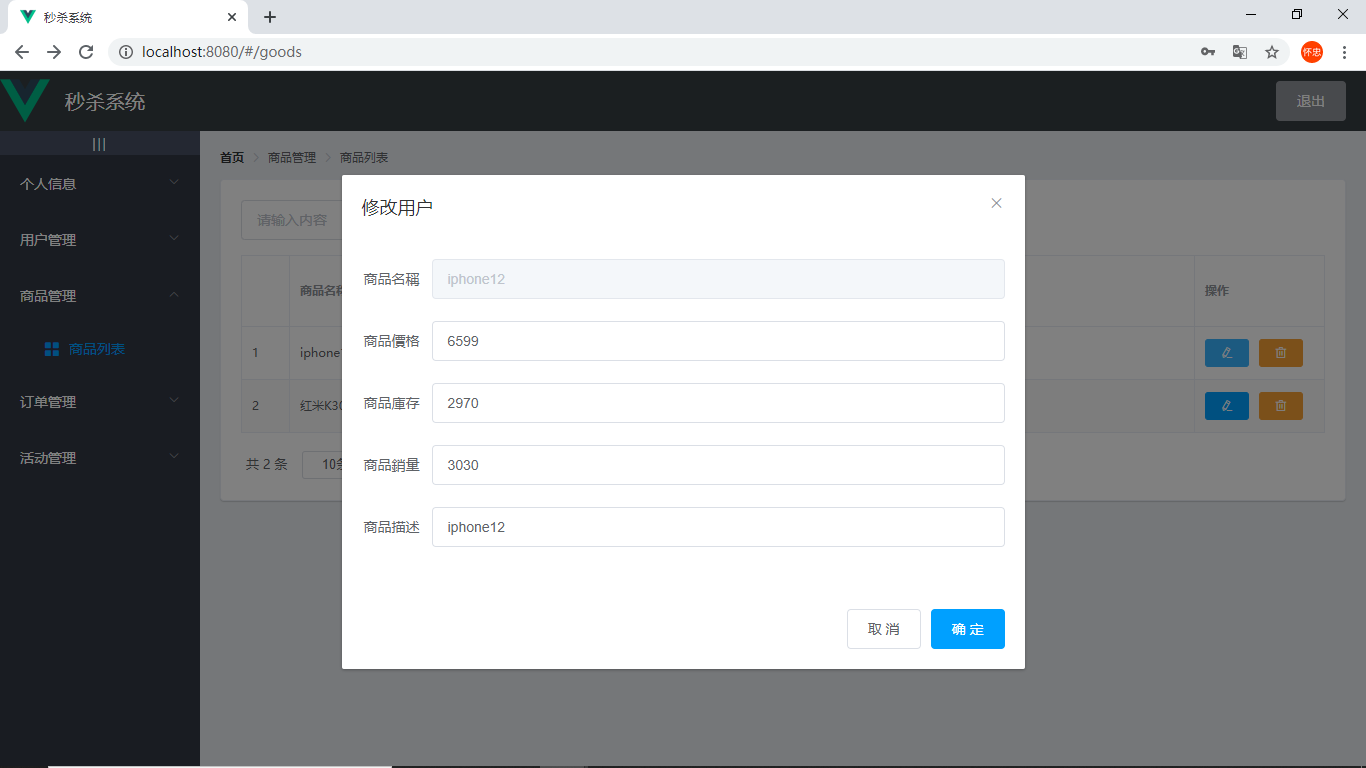




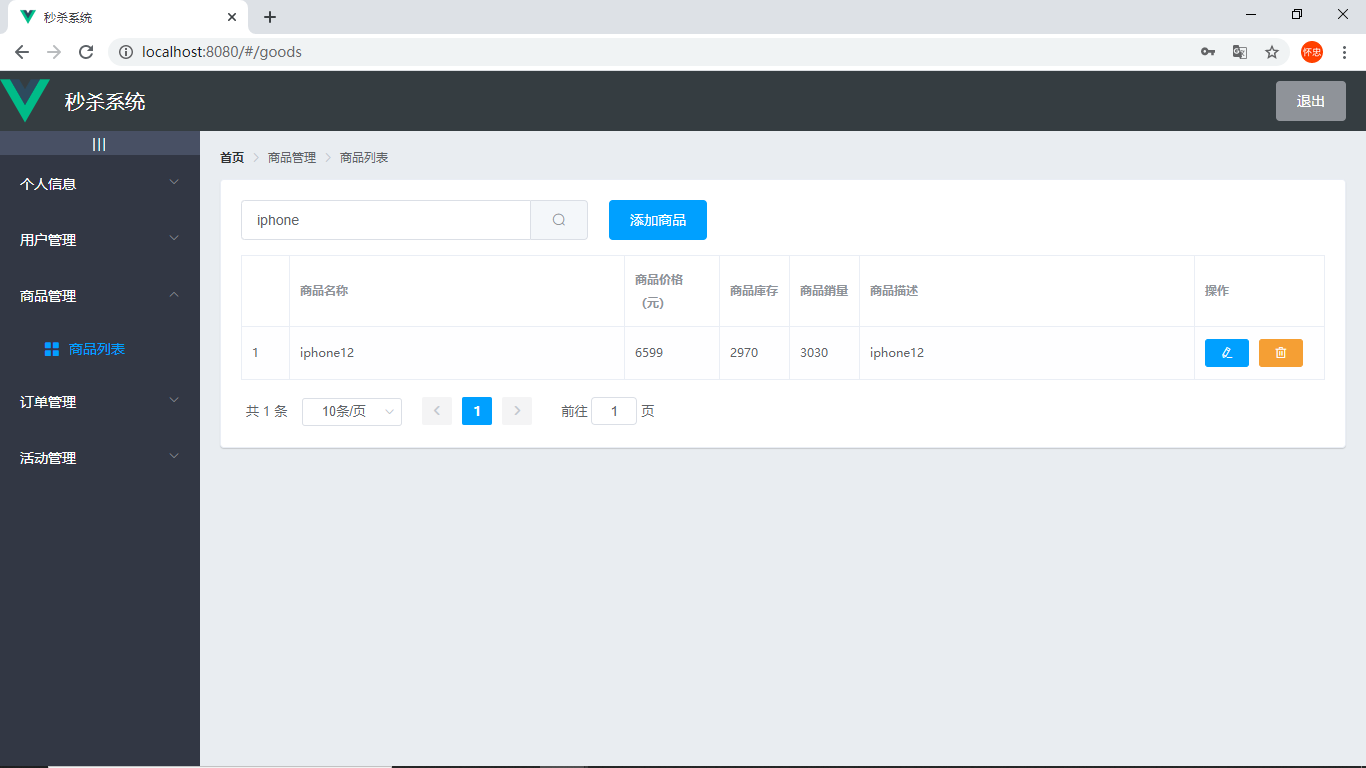
2、商品删除页面（删）



3、商品修改页面（改）

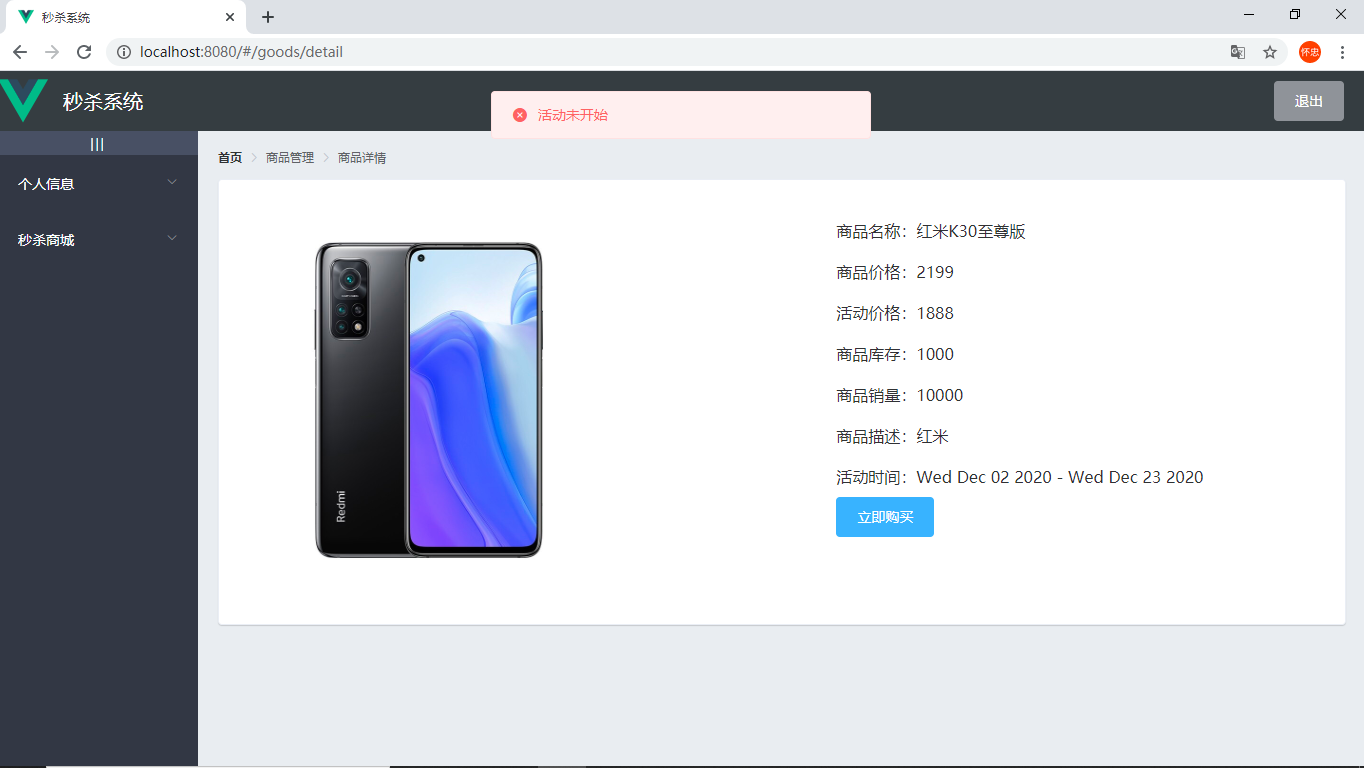


4、商品查询页面（查）

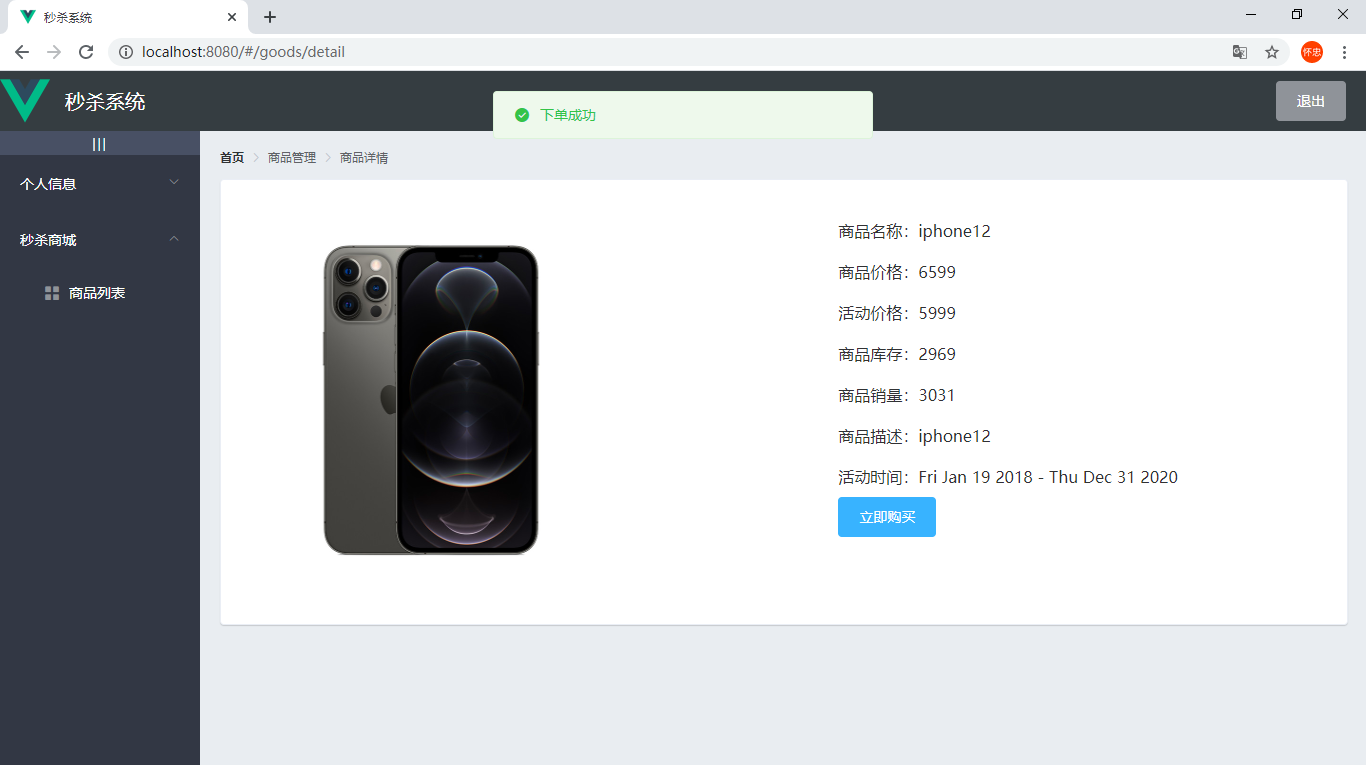


#### 商品秒杀模块的设计与实现

商品秒杀页面展示活动名称、价格、时间等基本信息，用户发起下单操作后，前端会校验其信息是否合法，例如，如果不在活动时间范围内，将提示用户活动未开始。或者库存已经耗尽，将提示用户库存不足。

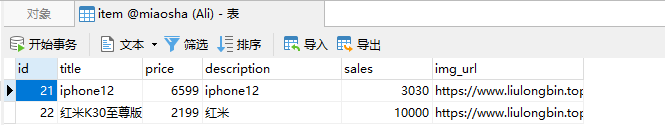


如果校验合法，前端首先访问/order/secKillToken接口，获取一个秒杀令牌，然后持此令牌向/order/createOrder接口发起真正的下单请求。在所有合法用户中，只有少量用户能够获得令牌，在订单生成模块中，服务器端后依次进行入参校验、用户是否合法、秒杀令牌是否合法等，然后尝试扣减库存。如果库存充足，则生成订单。为了维护订单号的有序性，在数据库中维护一个序列号，用于生成32位的订单号（例如2020122700020700）。接着依次写入订单、更新商品销量，生成日志流水等。整个交易模块作为一个事务执行，如果任一环节出现异常，回滚事务。如果事务提交成功，则提示用户下单成功。



## 性能优化策略及实现

1、库存信息分表存储

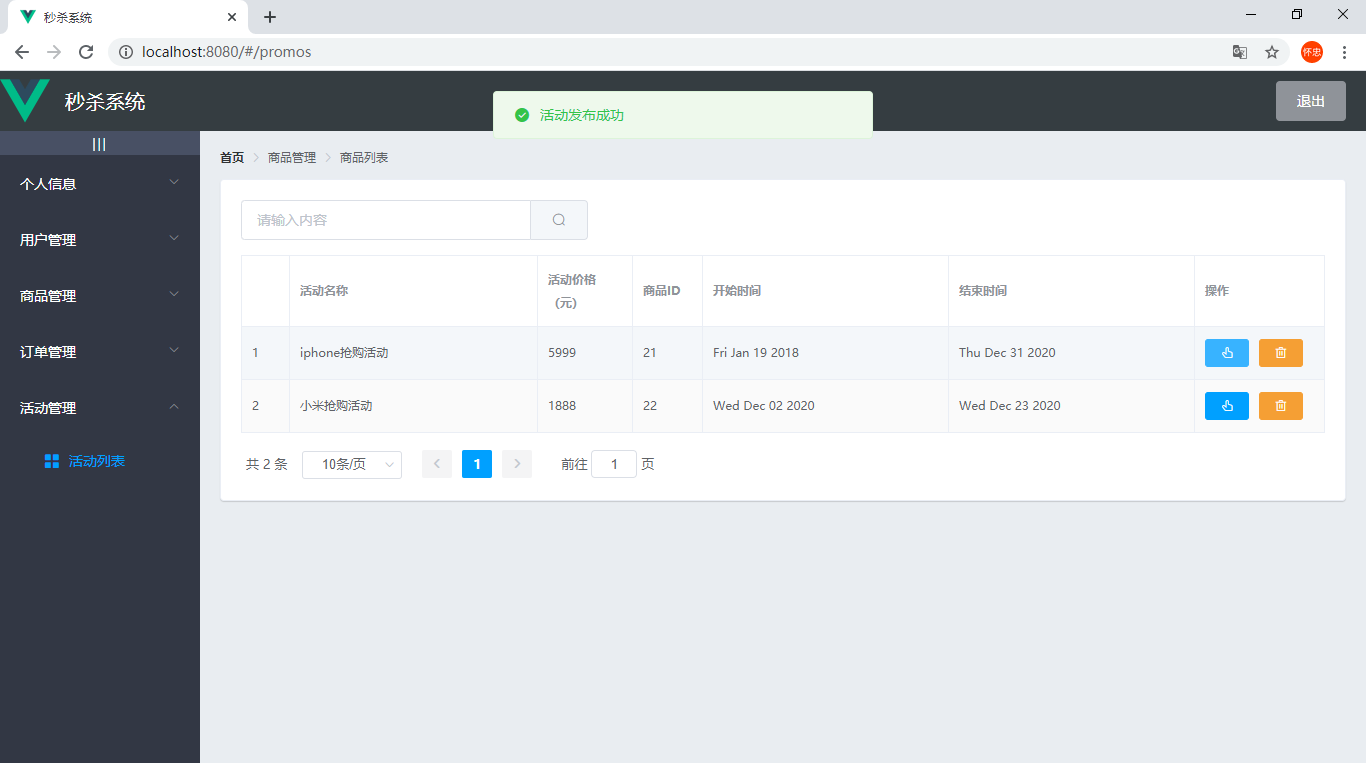




在秒杀业务中，商品库存项是热点数据，也是并发请求的竞争项。为了减轻服务端的压力，将商品普通信息与库存信息分离，避免读请求与写请求的竞争。

2、Redis缓存

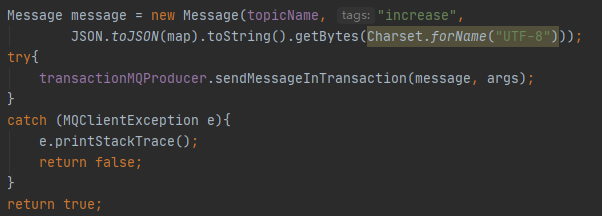
秒杀开启后，系统的性能瓶颈是库存的扣取。即购买同一商品的大量请求，均会访问该商品对应的库存项，并为其添加行锁。锁资源的请求与释放会消耗大量的CPU资源，减缓用户请求的相应时间，甚至引起服务器宕机。因此，在活动开启前，运维人员在后台管理程序中，将活动对应的商品库存写入Redis中，下单操作直接在缓存中扣取库存，由于Redis数据存储在内存中，减少了磁盘的IO操作，同时没有锁资源的消耗，系统性能大大提升。





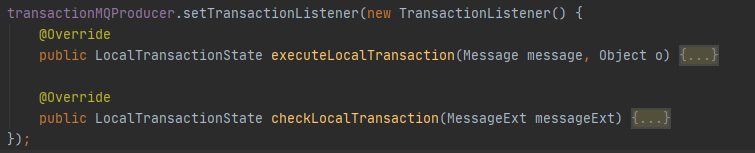
3、Rocket消息队列

引入Redis后，首先要解决的便是缓存与数据库的一致性问题。显然，服务端直接扣取数据库是不可取的。如果发起数据库扣减请求后，等待响应，则缓存失去意义。如果不等待响应，万一数据库出现回滚，则会导致缓存与数据库的不一致。因此引入RocketMQ消息队列，异步扣取数据库库存，达到维持缓存与数据库一致性的目的。



4、事务型消息与库存流水

整个秒杀交易过程应被当作一个事务整体来看待。因此，在消息队列端也引入事务型消息。同时，为了保持服务端事务与消息队列事务的整体性，引入库存流水表记录商品库存扣减操作的过程。



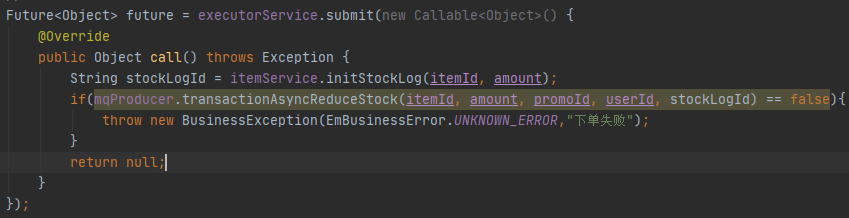
5、秒杀令牌构建秒杀大闸

在秒杀业务中，库存的数量是远远小于请求的并发数的。同时，库存又必须保证精准。因此我在库存之上，构建了秒杀令牌层，其数量是库存是3~5倍。用户首先竞争秒杀令牌，拥有秒杀令牌的用户才能继续竞争库存。由于秒杀令牌不需要过于精准，因此只在Redis中存储，从而省出了数据库和消息队列的操作，因此速度极快，同时又能达到很好的限制并发的作用。



6、队列化泄洪

秒杀系统的瓶颈在消息队列与数据库的更新。但是尽管做了上述的优化，其处理信息的能力仍然是有限的。因此这里采用队列话泄洪的方式，下游业务堆积而引起的服务器宕机。具体做法是构建一个拥塞窗口为20的线程池，用于真正的下单操作。当大量并发请求到达时，只保留20个线程进行业务操作，从而大量线程的锁竞争，迟滞服务端信息处理速度。

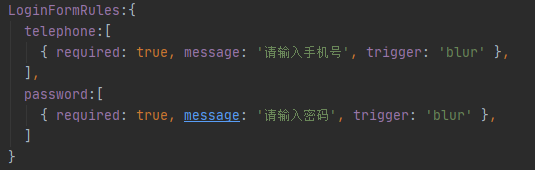


## 应用安全策略及实现

1. 入参校验

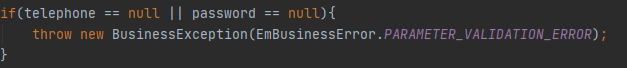
以用户登录过程为例介绍入参校验过程

前端首先会校验用户手机号与密码格式问题，如果不合法，提示用户。





服务端接收数据后，会再次验证，防止SQL注入。

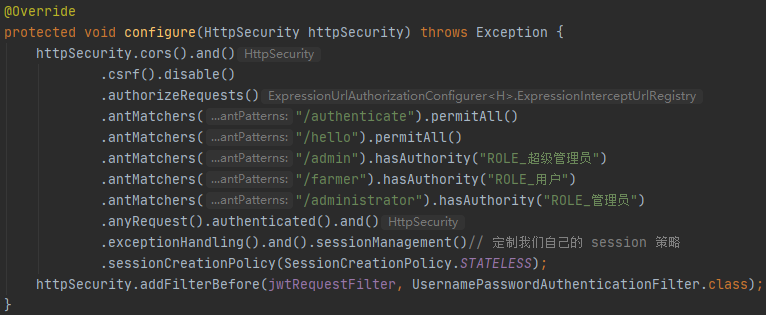


Model层会持续对用户信息进行校验



1. 权限管理

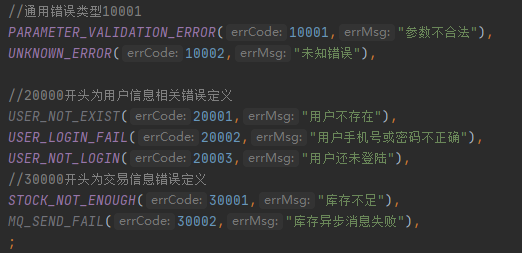
使用Spring Security框架进行权限管理。



1. 统一异常处理



1. 统一数据返回



## 项目部署

1. Vue

npm run install

npm run build

1. Java

cd target

java -jar server.jar

nohup java -jar server.jar >log/java.log &

1. MySQL

miaosha.sql

1. Redis

systemctl start redis

1. RocketMQ

nohup sh bin/mqnamesrv &

nohup sh bin/mqbroker -n localhost:9876 -c conf/broker.conf autoCreateTopicEnable=true &