**题目：小型库存预订模块（使用Django的orm方法进行设计）**

**需求1：**能根据商品 ID 查询当前库存，并提供根据商品名称进行搜索的功能

**操作步骤：**用户在搜索框中输入商品名称时，可能匹配到多个商品 ID，需要一起返回或分页返回

**考虑问题：**考虑性能及响应速度的情况下，如何设计数据库查询及缓存策略？

**已实现功能：**支持按商品名称搜索，分页显示搜索结果，实时显示库存数量

**具体实现：**

1. 为数据库查询设置索引：在 Product表中为name字段创建索引来加快按商品名称搜索的速度



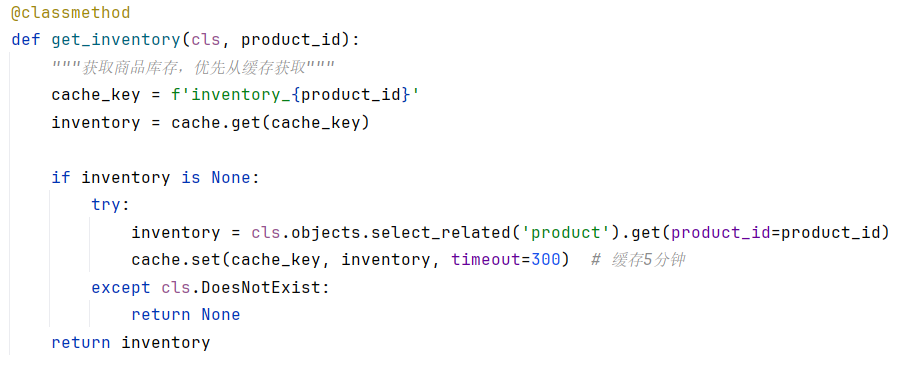
1. 在search\_products视图中使用filter方法来进行模糊查询，这个查询会使用到上面创建的索引，并在搜索视图中实现了两级缓存。



1. 缓存策略：使用Redis作为缓存后端，在settings.py中配置如下，并在库存查询中也实现缓存。



1. 在搜索视图和库存查询中实现缓存。



**需求2：**当外部系统请求预订商品（下单）时，若库存足够则扣减并返回成功，否则返回失败。需要考虑高并发场景下如何避免超卖。

**已实现功能：**使用数据库事务和行级锁确保并发安全，库存不足时自动禁用购买按钮，实时更新库存显示

**具体实现：**

1. 事务隔离：使用transaction.atomic()确保整个预订过程是原子的。要么全部成功，要么全部失败，不会出现部分成功的情况。
2. 行级锁机制：使用select\_for\_update()在查询时获取行级锁，确保同一时间只有一个事务可以修改特定商品的库存，其他事务必须等待当前事务完成。
3. 原子更新：使用Django的F()表达式进行原子更新，避免在应用层计算库存，防止并发问题，直接在数据库层面进行库存扣减。
4. 缓存一致性：在库存更新后立即删除相关缓存，确保下次查询时获取最新的库存数据，防止缓存数据与数据库不一致。
5. 添加异常处理：处理商品不存在的情况和库存不足的情况，以此确保系统在异常情况下也能正常工作。



**需求3：**为提高查询效率，可采用缓存机制；当库存变动时，需考虑如何更新或失效缓存，确保数据库与缓存一致性。

**已实现功能：**使用Redis缓存搜索结果，缓存商品库存信息，库存更新时自动清除相关缓存

**具体实现：**

1. 缓存更新策略：采用“先更新数据库，再删除缓存”的策略，使用缓存删除而不是更新，避免缓存不一致，并设置合理的缓存过期时间（5分钟）。
2. 缓存一致性保证：在库存更新时立即删除相关缓存，使用缓存过期时间作为最终一致性保证，对不存在的商品也进行缓存，避免缓存穿透。



**需求4：**异常情况：库存不足、数据库连接异常、缓存异常等，需要如何处理并保证系统的稳定。

**已实现功能：**处理库存不足情况，处理数据库连接异常，处理缓存异常，完整的日志记录

**具体实现：**

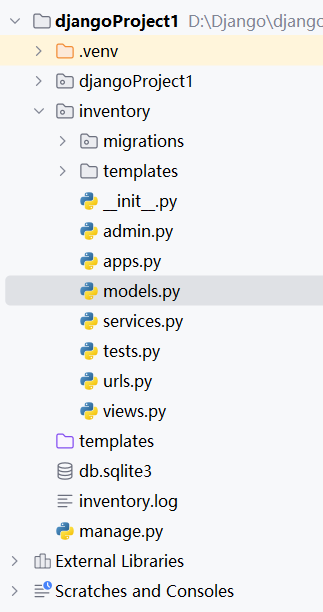
1. 分层异常处理：模型层：处理数据库操作异常；视图层：处理请求参数和业务逻辑异常；缓存层：处理缓存操作异常。
2. 异常降级处理：缓存异常时降级到数据库查询，数据库异常时返回友好的错误信息，参数错误时返回具体的错误提示。
3. 事务保证：使用数据库事务确保数据一致性，异常时自动回滚事务，保证库存操作的原子性。
4. 日志记录：记录详细的异常信息，便于问题排查和系统监控，区分不同级别的异常。



**需求5：**面向对象设计：简单分层，涵盖数据库访问层、缓存管理、业务服务等。

**具体实现：**

项目结构：



数据库访问层：



缓存管理层：



服务层：



视图层：

