

Họ tên: Hoàng Thị Mai Loan

MSV: B21DCCN490

CHAPTER 1: REQUIREMENT OF E-COMMERCE

1.1. Determine Requirements

1.1.1. Actors

Hệ thống bao gồm các tác nhân (actors) chính tương tác trực tiếp hoặc gián tiếp với hệ thống như sau:

- **User (Khách hàng):** Người dùng cuối của hệ thống. Họ có thể duyệt sản phẩm, thêm sản phẩm vào giỏ hàng, thực hiện thanh toán, theo dõi đơn hàng và cập nhật thông tin tài khoản cá nhân. Người dùng cũng có thể để lại đánh giá và phản hồi cho các sản phẩm đã mua.
- **Admin (Quản trị viên):** Người chịu trách nhiệm quản lý toàn bộ hệ thống. Admin có thể thêm, sửa, xoá sản phẩm; quản lý đơn hàng; kiểm soát tài khoản người dùng; xử lý khiếu nại; và giám sát các hoạt động hệ thống nhằm đảm bảo hoạt động ổn định và hiệu quả.
- **Payment Gateway (Cổng thanh toán):** Là dịch vụ của bên thứ ba chịu trách nhiệm xử lý các giao dịch thanh toán trực tuyến. Hệ thống sẽ kết nối với cổng thanh toán để thực hiện các giao dịch một cách an toàn và nhanh chóng, hỗ trợ nhiều hình thức thanh toán như thẻ tín dụng, ví điện tử, chuyển khoản ngân hàng, v.v.
- **Delivery Service (Dịch vụ giao hàng):** Là bên thứ ba cung cấp dịch vụ vận chuyển đơn hàng đến người dùng. Hệ thống sẽ tích hợp với dịch vụ giao hàng để tạo đơn vận chuyển, theo dõi trạng thái giao hàng, cập nhật thông tin giao hàng và xử lý các sự cố giao nhận nếu có.
- **Authentication Service (Dịch vụ xác thực):** Đảm bảo quá trình đăng nhập, đăng ký và xác minh danh tính người dùng. Có thể sử dụng dịch vụ xác thực nội bộ hoặc của bên thứ ba như OAuth (Google, Facebook), để bảo đảm an toàn thông tin và ngăn chặn truy cập trái phép.

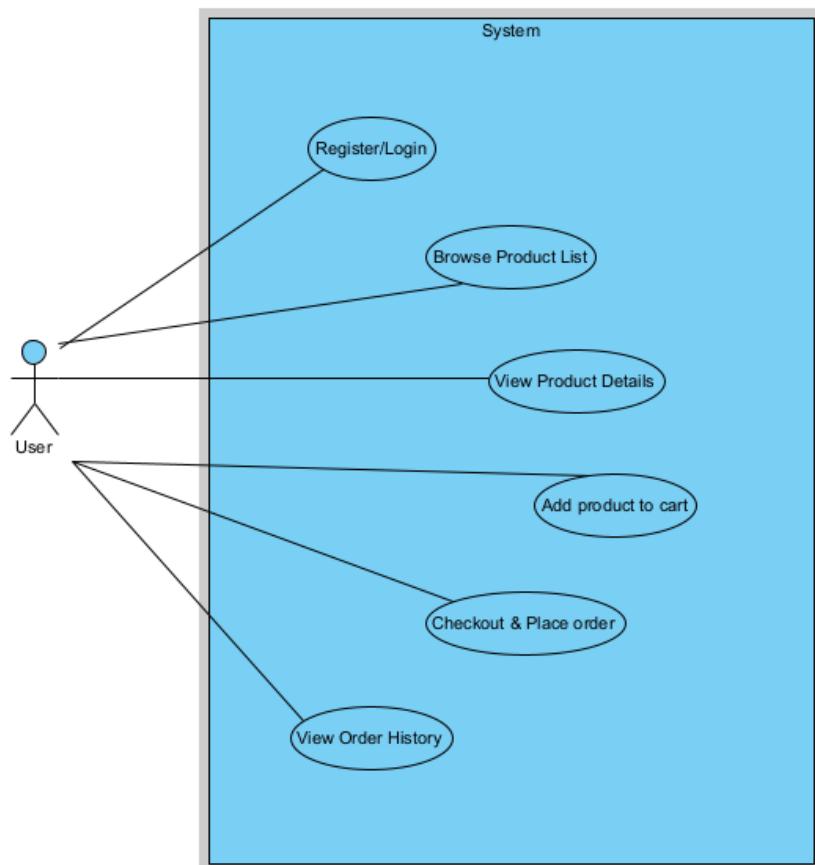
1.1.2. Functions with respect to actors

Actors	Functions
User	Đăng ký, đăng nhập
	Tìm kiếm, duyệt sản phẩm
	Đặt hàng, thanh toán
	Theo dõi đơn hàng
Admin	Quản lý sản phẩm, đơn hàng, người dùng
	Xử lý báo cáo doanh thu, đơn hàng
Payment Gateway	Xử lý thanh toán đơn hàng
Delivery Service	Cập nhật trạng thái vận chuyển
Authentication Service	Xác thực tài khoản người dùng

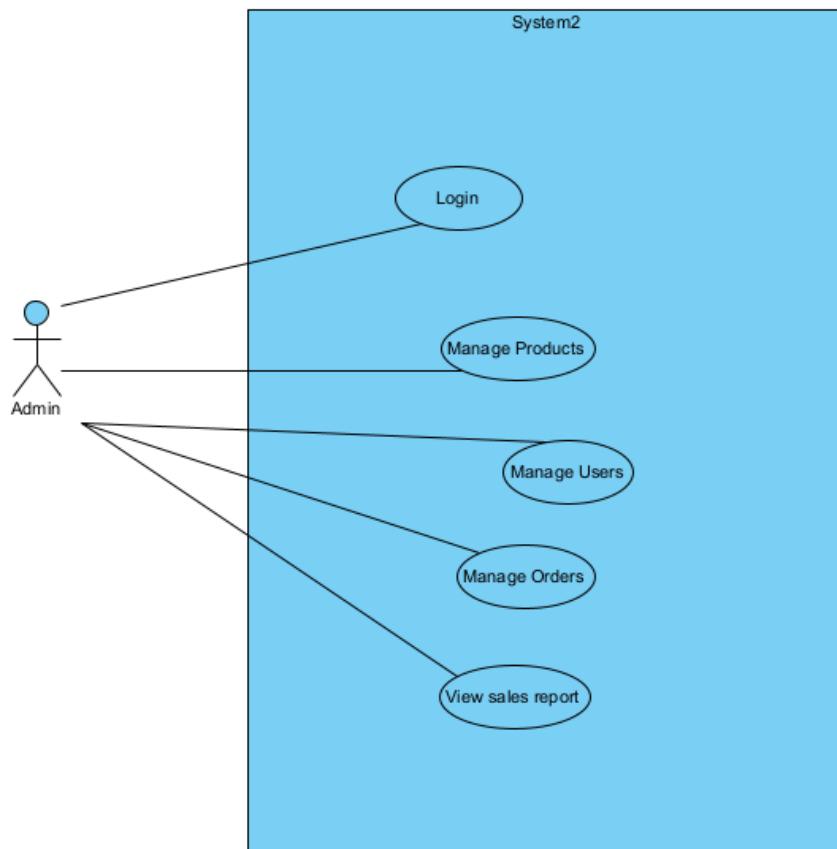
1.1.3. Use case diagrams

Hệ thống sẽ có các Use Case chính:

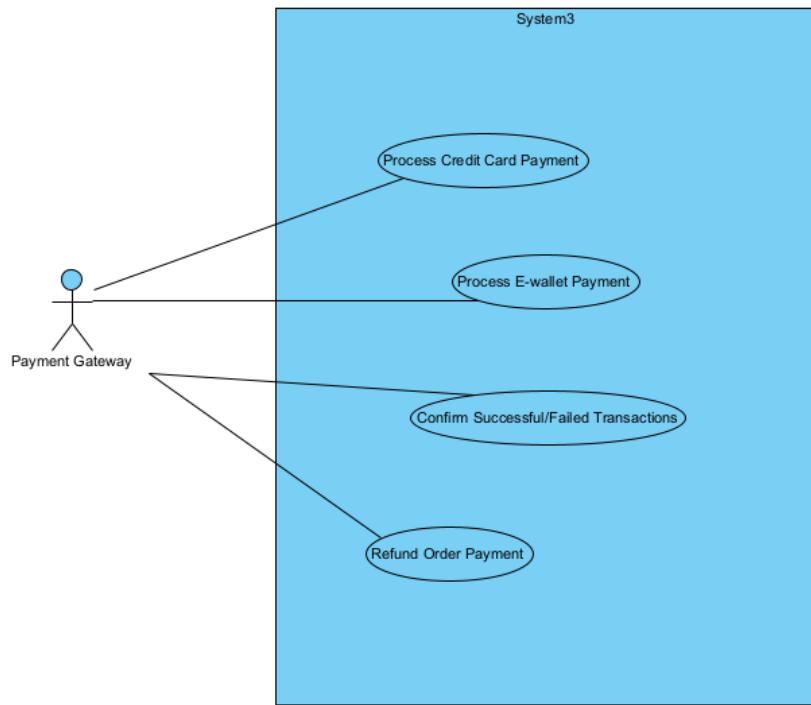
- **User Use Cases:** Đăng nhập, đăng ký, đặt hàng, xem đơn hàng.



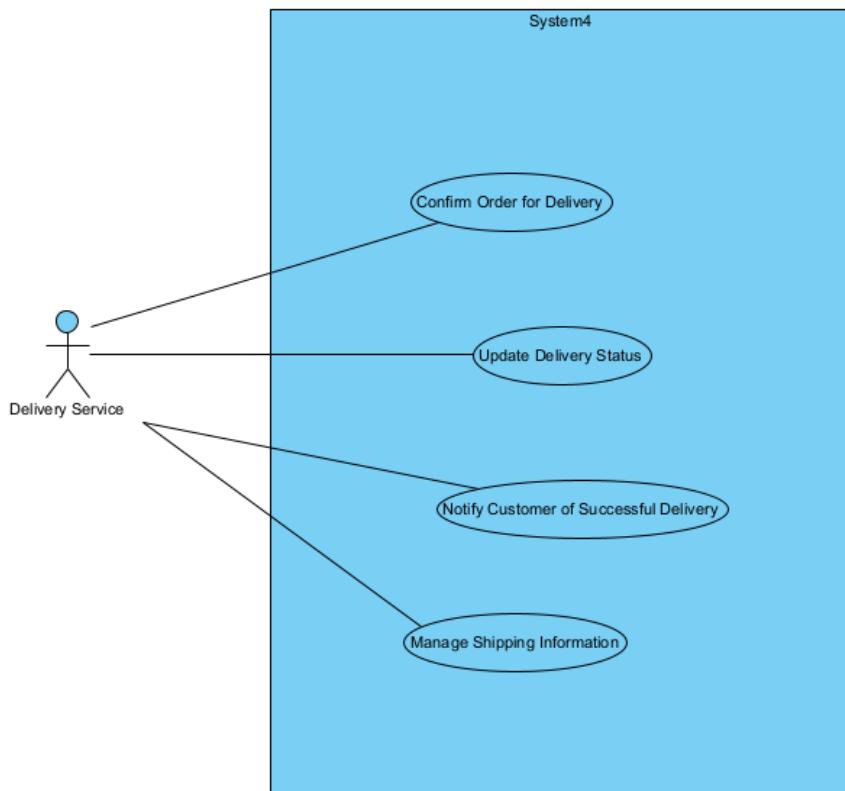
- **Admin Use Cases:** Quản lý sản phẩm, quản lý đơn hàng, quản lý người dùng.



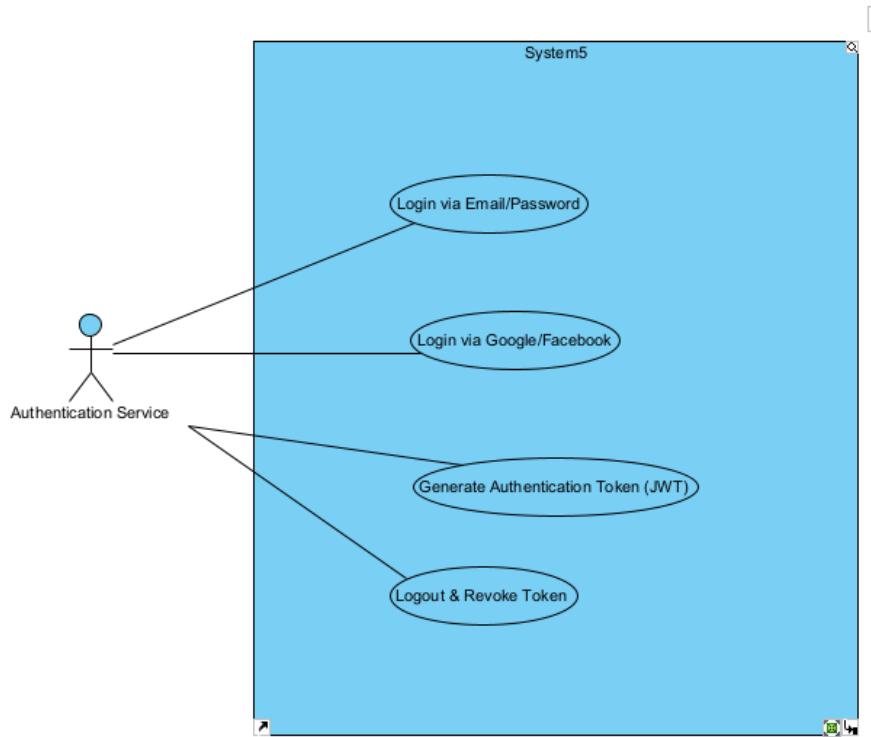
- **Payment Use Cases:** Xử lý thanh toán.



- **Delivery Use Cases:** Theo dõi trạng thái giao hàng.



- **Authentication Use Cases:** Xác thực người dùng.



1.2. Analyze Requirements

1.2.1. Decompose the system in microservices with Django

Hệ thống sẽ được chia thành các **microservices** độc lập:

1. **User Service**: Quản lý người dùng, đăng nhập, đăng ký, xác thực.
2. **Product Service**: Quản lý danh sách sản phẩm, danh mục.
3. **Order Service**: Xử lý đặt hàng, theo dõi trạng thái đơn hàng.
4. **Payment Service**: Xử lý thanh toán đơn hàng.
5. **Delivery Service**: Quản lý vận chuyển và cập nhật trạng thái đơn hàng.

1.2.2. Classes with attributes of service models (models)

Mỗi microservice có các models chính:

```
class User(models.Model):
```

```
    id = models.AutoField(primary_key=True)
```

```
    username = models.CharField(max_length=255, unique=True)
```

```
    email = models.EmailField(unique=True)
```

```
    password = models.CharField(max_length=255)
```

```
is_admin = models.BooleanField(default=False)
```

```
class Product(models.Model):
```

```
    id = models.AutoField(primary_key=True)
```

```
    name = models.CharField(max_length=255)
```

```
    price = models.DecimalField(max_digits=10, decimal_places=2)
```

```
    stock = models.IntegerField()
```

```
class Order(models.Model):
```

```
    id = models.AutoField(primary_key=True)
```

```
    user = models.ForeignKey(User, on_delete=models.CASCADE)
```

```
    total_price = models.DecimalField(max_digits=10, decimal_places=2)
```

```
    status = models.CharField(max_length=20, choices=[('Pending', 'Pending'), ('Shipped', 'Shipped'), ('Delivered', 'Delivered')])
```

```
class Payment(models.Model):
```

```
    id = models.AutoField(primary_key=True)
```

```
    order = models.ForeignKey(Order, on_delete=models.CASCADE)
```

```
    payment_status = models.CharField(max_length=20, choices=[('Pending', 'Pending'), ('Completed', 'Completed'), ('Failed', 'Failed')])
```

```
class Delivery(models.Model):
```

```
    id = models.AutoField(primary_key=True)
```

```
    order = models.ForeignKey(Order, on_delete=models.CASCADE)
```

```
    delivery_status = models.CharField(max_length=20, choices=[('Pending', 'Pending'), ('In Transit', 'In Transit'), ('Delivered', 'Delivered')])
```

1.2.3. Determine functions in services (views)

Mỗi service sẽ có các views xử lý logic nghiệp vụ:

User Service (views.py)

```
def register(request):
    # Xử lý đăng ký người dùng
def login(request):
    # Xử lý đăng nhập người dùng
```

Product Service (views.py)

```
def get_products(request):
    # Lấy danh sách sản phẩm
def get_product_detail(request, product_id):
    # Lấy thông tin chi tiết sản phẩm
```

Order Service (views.py)

```
def create_order(request):
    # Tạo đơn hàng
def get_order_status(request, order_id):
    # Xem trạng thái đơn hàng
```

Payment Service (views.py)

```
def process_payment(request, order_id):
    # Xử lý thanh toán đơn hàng
```

Delivery Service (views.py)

```
def track_order(request, order_id):
    # Theo dõi trạng thái vận chuyển
```

1.2.4. Determine templates

Mỗi dịch vụ có giao diện riêng:

User Service

- register.html: Trang đăng ký tài khoản.
- login.html: Trang đăng nhập.
- profile.html: Trang quản lý thông tin cá nhân.

Product Service

- product_list.html: Danh sách sản phẩm.
- product_detail.html: Trang chi tiết sản phẩm.

Order Service

- order_history.html: Lịch sử đơn hàng của người dùng.
- order_detail.html: Chi tiết đơn hàng.

Payment Service

- payment_confirmation.html: Xác nhận thanh toán.
- payment_status.html: Trạng thái giao dịch.

Delivery Service

- tracking.html: Theo dõi đơn hàng theo mã vận đơn.
- delivery_status.html: Trạng thái đơn hàng.

1.2.5. Determine REST API connecting services

Các API REST chính giữa các dịch vụ:

User Service

POST /api/user/register
POST /api/user/login
GET /api/user/profile

Product Service

GET /api/product/list
GET /api/product/{id}
POST /api/product/create

Order Service

POST /api/order/create
GET /api/order/{id}

Payment Service

POST /api/payment/process/{order_id}
GET /api/payment/status/{order_id}

Delivery Service

GET /api/delivery/track/{order_id}

POST /api/delivery/update/{order_id}

1.3. Conclusion

Hệ thống microservice với Django được thiết kế theo hướng phân tán, mỗi dịch vụ có chức năng riêng biệt và giao tiếp với nhau thông qua API REST. Mô hình này giúp hệ thống dễ mở rộng, bảo trì và đảm bảo hiệu suất cao khi vận hành thực tế.

CHAPTER 2: DESIGN E-COMMERCE SYSTEM WITH MICROSERVICES AND DJANGO

2.1. Design services/components

2.2 Design classes and methods in component

2.3 Design API

B1: Tạo các Django Project cho mỗi dịch vụ

User service:

```
D:\assignment4>django-admin startproject user_service
D:\assignment4>cd user_service
D:\assignment4\user_service>django-admin startapp users
D:\assignment4\user_service>
```

Product service:

```
D:\assignment4>django-admin startproject product_service
D:\assignment4>cd product_service
D:\assignment4\product_service>django-admin startapp products
D:\assignment4\product_service>_
```

Order service:

```
D:\assignment4>django-admin startproject order_service
D:\assignment4>cd order_service
D:\assignment4\order_service>django-admin startapp orders
D:\assignment4\order_service>_
```

Payment service:

```
D:\assignment4>django-admin startproject payment_service
D:\assignment4>cd payment_service
D:\assignment4\payment_service>django-admin startapp payments
D:\assignment4\payment_service>_
```

Delivery service:

```
D:\assignment4>django-admin startproject delivery_service
D:\assignment4>cd delivery_service
D:\assignment4\delivery_service>django-admin startapp deliveries
D:\assignment4\delivery_service>_
```

B2: Thiết kế models cho mỗi service:

User model:

```
users > 🐍 models.py > User
1  from django.contrib.auth.models import AbstractUser
2  from django.db import models
3
4  class User(AbstractUser):
5      phone_number = models.CharField(max_length=15, unique=True)
6      address = models.TextField(null=True, blank=True)
7
```

Product model:

```
products > 🐍 models.py > ...
1  from django.db import models
2
3  class Category(models.Model):
4      name = models.CharField(max_length=255)
5
6  class Product(models.Model):
7      name = models.CharField(max_length=255)
8      description = models.TextField()
9      price = models.DecimalField(max_digits=10, decimal_places=2)
10     stock = models.IntegerField()
11     category = models.ForeignKey(Category, on_delete=models.CASCADE)
12
```

Order model:

```
orders > 🐍 models.py > ...
1  from django.db import models
2
3  class Order(models.Model):
4      user_id = models.IntegerField()
5      total_price = models.DecimalField(max_digits=10, decimal_places=2)
6      status = models.CharField(max_length=20, choices=[("Pending", "Pending"), ("Completed", "Completed")])
7      created_at = models.DateTimeField(auto_now_add=True)
8
9  class OrderDetail(models.Model):
10     order = models.ForeignKey(Order, on_delete=models.CASCADE)
11     product_id = models.IntegerField()
12     quantity = models.IntegerField()
13     price = models.DecimalField(max_digits=10, decimal_places=2)
```

Payment model:

```

payments > models.py > Payment
1  from django.db import models
2
3  class Payment(models.Model):
4      order_id = models.IntegerField()
5      payment_status = models.CharField(max_length=20, choices=[("Pending", "Pending"),
6          ("Completed", "Completed")])
7      payment_method = models.CharField(max_length=50)
8      transaction_id = models.CharField(max_length=255, unique=True, null=True, blank=True)

```

Delivery model:

```

deliveries > models.py > ...
1  from django.db import models
2
3  class Delivery(models.Model):
4      order_id = models.IntegerField()
5      delivery_status = models.CharField(max_length=20, choices=[("Pending", "Pending"),
6          ("Shipped", "Shipped"),
7          ("Delivered", "Delivered")])
8      tracking_number = models.CharField(max_length=255, unique=True)
9      estimated_delivery_date = models.DateField()
10

```

B3: Tạo serializer cho các model:

UserSerializer:

```

users > serializers.py > UserSerializer > Meta
1  from rest_framework import serializers
2  from .models import User
3
4  class UserSerializer(serializers.ModelSerializer):
5      class Meta:
6          model = User
7          fields = ['id', 'username', 'email', 'first_name',
8                  'last_name', 'phone_number', 'address']
9

```

ProductSerializer:

```

products > serializers.py > ...
1  from rest_framework import serializers
2  from .models import Product
3
4  class ProductSerializer(serializers.ModelSerializer):
5      class Meta:
6          model = Product
7          fields = '__all__' # Bao gồm tất cả các trường của model Product
8

```

OrderSerializer:

```
orders > serializers.py > OrderSerializer
1  from rest_framework import serializers
2  from .models import Order, OrderDetail
3
4  class OrderDetailSerializer(serializers.ModelSerializer):
5      order = serializers.PrimaryKeyRelatedField(queryset=Order.objects.all())
6
7      class Meta:
8          model = OrderDetail
9          fields = ['id', 'order', 'quantity', 'price']
10
11 class OrderSerializer(serializers.ModelSerializer):
12     class Meta:
13         model = Order
14         fields = ['id', 'total_price', 'status', 'created_at']
15
```

PaymentSerializer:

```
payments > serializers.py > ...
1  from rest_framework import serializers
2  from .models import Payment
3
4  class PaymentSerializer(serializers.ModelSerializer):
5      class Meta:
6          model = Payment
7          fields = '__all__' # Trả về tất cả các trường của Payment
8
```

DeliverySerializer:

```
deliveries > serializers.py > ...
1  from rest_framework import serializers
2  from .models import Delivery
3
4  class DeliverySerializer(serializers.ModelSerializer):
5      class Meta:
6          model = Delivery
7          fields = '__all__' # Trả về tất cả các trường của Delivery
8
```

B4: Viết view CRUD

User:

```
users > ⚡ views.py > ...
1  from rest_framework.generics import ListCreateAPIView
2  from .models import User
3  from .serializers import UserSerializer
4  from rest_framework.generics import RetrieveUpdateDestroyAPIView
5
6  class UserListCreateView(ListCreateAPIView):
7      queryset = User.objects.all()
8      serializer_class = UserSerializer
9
10
11 class UserDetailView(RetrieveUpdateDestroyAPIView):
12     queryset = User.objects.all()
13     serializer_class = UserSerializer
```

Product:

```
products > ⚡ views.py > 📄 ProductListCreateView > ⏺ perform_create
1  from rest_framework import generics
2  from .models import Product
3  from .serializers import ProductSerializer
4
5  class ProductListCreateView(generics.ListCreateAPIView):
6      queryset = Product.objects.all()
7      serializer_class = ProductSerializer
8
9      def perform_create(self, serializer):
10         # Tự động xử lý khi tạo mới sản phẩm
11         serializer.save()
12
13  class ProductRetrieveUpdateDestroyView(generics.RetrieveUpdateDestroyAPIView):
14      queryset = Product.objects.all()
15      serializer_class = ProductSerializer
16
```

Order:

```

orders > ✏ views.py > ...
  1 from rest_framework import status
  2 from rest_framework.generics import ListCreateAPIView, RetrieveUpdateDestroyAPIView
  3 from rest_framework.response import Response
  4 from .models import Order, OrderDetail
  5 from .serializers import OrderSerializer, OrderDetailSerializer
  6 import requests
  7
  8 # URL của các microservices
  9 USER_SERVICE_URL = "http://localhost:8001/api/users/"
10 PRODUCT_SERVICE_URL = "http://localhost:8002/api/products/"
11
12 class OrderListCreateView(ListCreateAPIView):
13     queryset = Order.objects.all()
14     serializer_class = OrderSerializer
15
16     def create(self, request, *args, **kwargs):
17         # Lấy thông tin user từ UserService
18         user_id = request.data.get('user_id')
19         user_response = requests.get(f"{USER_SERVICE_URL}{user_id}/")
20
21         if user_response.status_code != 200:
22             return Response({"detail": "User not found"}, status=status.HTTP_404_NOT_FOUND)
23
24         # Tạo đơn hàng
25         order = Order.objects.create(
26             user_id=user_id,
27             total_price=request.data.get('total_price'),
28             status=request.data.get('status'),
29             created_at=request.data.get('created_at')
30         )
31
32
33         # Sau khi tạo đơn hàng, tạo OrderDetail cho các sản phẩm
34         products = request.data.get('products', [])
35         for product in products:
36             product_response = requests.get(f"{PRODUCT_SERVICE_URL}{product['product_id']}/")
37             if product_response.status_code != 200:
38                 return Response({"detail": f"Product with id {product['product_id']} not found"}, status=200)
39             OrderDetail.objects.create(
40
41                 if product_response.status_code != 200:
42                     return Response({"detail": f"Product with id {product['product_id']} not found"}, status=status.HTTP_404_NOT_FOUND)
43
44                 OrderDetail.objects.create(
45                     order=order,
46                     product_id=product['product_id'],
47                     quantity=product['quantity'],
48                     price=product['price']
49
50             # Trả về thông tin đơn hàng
51             order_response = OrderSerializer(order).data
52             order_response['user'] = user_response.json() # Thêm thông tin người dùng
53             return Response(order_response, status=status.HTTP_201_CREATED)
54
55         def list(self, request, *args, **kwargs):
56             # Lấy danh sách tất cả các đơn hàng
57             orders = self.get_queryset()

```

```

55     # Lấy thông tin người dùng từ UserService cho từng đơn hàng
56     orders_data = []
57     for order in orders:
58         # Lấy thông tin người dùng từ UserService
59         user_response = requests.get(f"{USER_SERVICE_URL}{order.user_id}/")
60         if user_response.status_code == 200:
61             user_data = user_response.json()
62         else:
63             user_data = None # Nếu không tìm thấy người dùng, trả về None
64
65
66         # Chuyển dữ liệu đơn hàng thành dữ liệu muốn trả về
67         order_data = OrderSerializer(order).data
68         order_data['user'] = user_data # Thêm thông tin người dùng vào response
69         orders_data.append(order_data)
70
71     return Response(orders_data)
72

```

```

74 class OrderDetailListCreateView(ListCreateAPIView):
75     queryset = OrderDetail.objects.all()
76     serializer_class = OrderDetailSerializer
77
78     def create(self, request, *args, **kwargs):
79         # Kiểm tra xem Order có tồn tại không
80         order_id = request.data.get('order')
81         try:
82             order = Order.objects.get(id=order_id)
83         except Order.DoesNotExist:
84             return Response({"detail": "Order not found"}, status=status.HTTP_404_NOT_FOUND)
85
86         # Nếu Order hợp lệ, tiến hành tạo OrderDetail
87         return super().create(request, *args, **kwargs)
88
89     def list(self, request, *args, **kwargs):
90         # Lấy danh sách tất cả các chi tiết đơn hàng
91         order_details = self.get_queryset()
92

```

```

93     # Lấy thông tin chi tiết đơn hàng từ các API UserService và ProductService
94     order_details_data = []
95     for order_detail in order_details:
96         # Lấy thông tin đơn hàng từ Order
97         order = Order.objects.get(id=order_detail.order_id)
98
99         # Lấy thông tin sản phẩm từ ProductService
100        product_response = requests.get(f"{PRODUCT_SERVICE_URL}{order_detail.product_id}/")
101        if product_response.status_code == 200:
102            product_data = product_response.json()
103        else:
104            product_data = None # Nếu không tìm thấy sản phẩm, trả về None
105

```

```

105     # Chuyển dữ liệu chi tiết đơn hàng thành dữ liệu muốn trả về
106     order_detail_data = OrderDetailSerializer(order_detail).data
107     order_detail_data['order'] = order.id # Thêm thông tin đơn hàng (ID)
108     order_detail_data['product'] = product_data # Thêm thông tin sản phẩm vào response
109     order_details_data.append(order_detail_data)
110
111     return Response(order_details_data)
112
113
114
115 class OrderDetailView(RetrieveUpdateDestroyAPIView):
116     queryset = OrderDetail.objects.all()
117     serializer_class = OrderDetailSerializer
118
119     def retrieve(self, request, *args, **kwargs):
120         # Lấy chi tiết đơn hàng theo ID
121         order_detail = self.get_object()
122
123
124         # Lấy thông tin sản phẩm từ ProductService
125         product_response = requests.get(f"{PRODUCT_SERVICE_URL}{order_detail.product_id}/")
126         if product_response.status_code == 200:
127             order_detail.product_info = product_response.json() # Thêm thông tin sản phẩm vào kết quả trả về
128
129         # Trả về kết quả
130         serializer = self.get_serializer(order_detail)
131         order_detail_data = serializer.data
132         order_detail_data['product'] = product_response.json() # Thêm thông tin sản phẩm
133
134         return Response(order_detail_data)

```

Payment:

```

payments > views.py > PaymentListCreateView > list
1 import requests
2 from rest_framework import generics
3 from rest_framework.response import Response
4 from .models import Payment
5 from .serializers import PaymentSerializer
6
7 # URL của OrderService để lấy thông tin đơn hàng
8 ORDER_SERVICE_URL = "http://localhost:8003/api/orders/"
9
10 class PaymentListCreateView(generics.ListCreateAPIView):
11     queryset = Payment.objects.all()
12     serializer_class = PaymentSerializer
13
14     def list(self, request, *args, **kwargs):
15         payments = self.get_queryset()
16         payments_data = []
17
18         for payment in payments:
19             # Lấy thông tin đơn hàng từ OrderService
20             order_response = requests.get(f"{ORDER_SERVICE_URL}{payment.order_id}/")
21             order_data = order_response.json() if order_response.status_code == 200 else
22

```

```

22         # Chuyển dữ liệu thanh toán thành dữ liệu trả về
23         payment_data = PaymentSerializer(payment).data
24         payment_data['order'] = order_data # Thêm thông tin đơn hàng
25         payments_data.append(payment_data)
26
27     return Response(payments_data)
28
29
30 class PaymentRetrieveUpdateDestroyView(generics.RetrieveUpdateDestroyAPIView):
31     queryset = Payment.objects.all()
32     serializer_class = PaymentSerializer
33

```

Delivery:

```

deliveries > views.py > DeliveryListCreateView > list
1 import requests
2 from rest_framework import generics
3 from rest_framework.response import Response
4 from .models import Delivery
5 from .serializers import DeliverySerializer
6
7 # URL của OrderService để lấy thông tin đơn hàng
8 ORDER_SERVICE_URL = "http://localhost:8003/api/orders/"
9
10 class DeliveryListCreateView(generics.ListCreateAPIView):
11     queryset = Delivery.objects.all()
12     serializer_class = DeliverySerializer
13
14     def list(self, request, *args, **kwargs):
15         deliveries = self.get_queryset()
16         deliveries_data = []
17
18         for delivery in deliveries:
19             # Lấy thông tin đơn hàng từ OrderService
20             order_response = requests.get(f"{ORDER_SERVICE_URL}{delivery.order_id}/")
21             order_data = order_response.json() if order_response.status_code == 200 else
22

```

```

22
23         # Chuyển dữ liệu giao hàng thành dữ liệu trả về
24         delivery_data = DeliverySerializer(delivery).data
25         delivery_data['order'] = order_data # Thêm thông tin đơn hàng
26         deliveries_data.append(delivery_data)
27
28     return Response(deliveries_data)
29
30 class DeliveryRetrieveUpdateDestroyView(generics.RetrieveUpdateDestroyAPIView):
31     queryset = Delivery.objects.all()
32     serializer_class = DeliverySerializer
33

```

Cấu hình URL:

User:

```
user_service > urls.py > ...
15     2. Add a URL to urlpatterns:  path('blog/', include('blog.
16     """
17     from django.contrib import admin
18     from django.urls import path, include
19
20     urlpatterns = [
21         path('admin/', admin.site.urls),
22         path('api/users/', include('users.urls')),
23     ]
24
```

```
products > urls.py > ...
1  from django.urls import path
2  from .views import ProductListCreateView, ProductRetrieveUpdateDestroyView
3
4  urlpatterns = [
5      path('', ProductListCreateView.as_view(), name='product-list-create'),
6      path('<int:pk>/', ProductRetrieveUpdateDestroyView.as_view(), name='product-detail'),
7  ]
8
```

Product:

```
product_service > urls.py > ...
15     2. Add a URL to urlpatterns:  path('blog/', include('blog.
16     """
17     from django.contrib import admin
18     from django.urls import path, include
19
20     urlpatterns = [
21         path('admin/', admin.site.urls),
22         path('api/products/', include('products.urls')),
23     ]
24
```

```
products > urls.py > ...
1  from django.urls import path
2  from .views import ProductListCreateView, ProductRetrieveUpdateDestroyView
3
4  urlpatterns = [
5      path('', ProductListCreateView.as_view(), name='product-list-create'),
6      path('<int:pk>/', ProductRetrieveUpdateDestroyView.as_view(), name='product-detail'),
7  ]
8
```

Order:

```
order_service > ✎ urls.py > ...
16 """
17 from django.contrib import admin
18 from django.urls import path, include
19
20 urlpatterns = [
21     path('admin/', admin.site.urls),
22     path('api/orders/', include('orders.urls')),
23
24 ]
25
```

```
orders > ✎ urls.py > ...
1 from django.urls import path
2 from .views import OrderListCreateView, OrderDetailView, OrderDetailListCreateView
3
4 urlpatterns = [
5     # API cho các đơn hàng
6     path('', OrderListCreateView.as_view(), name='order-list'), # GET (list) & POST (cre
7     path('<int:pk>/', OrderDetailView.as_view(), name='order-detail'), # GET, PUT, DELET
8
9     # API cho các chi tiết đơn hàng
10    path('order-details/', OrderDetailListCreateView.as_view(), name='order-detail-list-c
11 ]
12
```

Payment:

```
payment_service > ✎ urls.py > ...
• 16 """
17 from django.contrib import admin
18 from django.urls import path, include
19
20 urlpatterns = [
21     path('admin/', admin.site.urls),
22     path('api/payments/', include('payments.urls')),
23
24 ]
```

```
payments > ✎ urls.py > ...
1 from django.urls import path
2 from .views import PaymentListCreateView, PaymentRetrieveUpdateDestroyView
3
4 urlpatterns = [
5     path('', PaymentListCreateView.as_view(), name='payment-list'),
6     path('<int:pk>/', PaymentRetrieveUpdateDestroyView.as_view(), name='payment-detail'),
7 ]
8
```

Delivery:

```
delivery_service > urls.py > ...
17 from django.contrib import admin
18 from django.urls import path, include
19
20 urlpatterns = [
21     path('admin/', admin.site.urls),
22     path('api/deliveries/', include('deliveries.urls')),
23
24 ]
25
```

```
deliveries > urls.py > ...
1 from django.urls import path
2 from .views import DeliveryListCreateView, DeliveryRetrieveUpdateDestroyView
3
4 urlpatterns = [
5     path('', DeliveryListCreateView.as_view(), name='delivery-list'),
6     path('<int:pk>/', DeliveryRetrieveUpdateDestroyView.as_view(), name='delivery-detail')
7 ]
```

Run các service:

User: **python manage.py runserver 8001**

Product: **python manage.py runserver 8002**

Order: **python manage.py runserver 8003**

Payment: **python manage.py runserver 8004**

Delivery: **python manage.py runserver 8005**

Test các API:

User:

My Workspace

assignment4

- products
 - POST add product
 - GET get all products
 - GET get product by id
 - PUR update product
 - DEL delete product
 - users
 - POST add user
 - GET get all users
 - GET get user by id
 - PUR update user
 - DEL delete user
 - orders
 - POST create order
 - POST create order detail
 - GET get all orders
 - GET get all order details
 - GET get order by id
 - payments
 - POST create payment
 - PUR update payment
 - GET get all payments
 - GET get payment by id
 - DEL delete payment
 - deliveries
 - POST create delivery
 - PUR update delivery
 - GET get all deliveries
 - GET get delivery by id
 - DEL delete delivery

assignment4 / users get all users

GET http://localhost:8001/api/users/

Params Authorization Headers Body Scripts Settings

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (10) Test Results

{ JSON } Preview Visualize

```
[{"id": 1, "username": "ducanh2111", "email": "ducanh@gmail.com", "first_name": "Duc", "last_name": "Anh", "phone_number": "0987654321", "address": "Hanoi"}]
```

200 OK 0 ms - 474 B Save Response

Postbot Runner Start Proxy Cookies Vault Trash 3:39 PM 2/04/2025

My Workspace

assignment4

- products
 - POST add product
 - GET get all products
 - GET get product by id
 - PUR update product
 - DEL delete product
 - users
 - POST add user
 - GET get all users
 - GET get user by id
 - PUR update user
 - DEL delete user
 - orders
 - POST create order
 - POST create order detail
 - GET get all orders
 - GET get all order details
 - GET get order by id
 - payments
 - POST create payment
 - PUR update payment
 - GET get all payments
 - GET get payment by id
 - DEL delete payment
 - deliveries
 - POST create delivery
 - PUR update delivery
 - GET get all deliveries
 - GET get delivery by id
 - DEL delete delivery

assignment4 / users add user

POST http://localhost:8001/api/users/

Params Authorization Headers (9) Body Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
{"username": "theanh", "email": "theanh@gmail.com", "first_name": "The", "last_name": "Anh", "phone_number": "0987654321", "address": "Hanoi"}
```

Body Cookies Headers (10) Test Results

{ JSON } Preview Visualize

```
{"id": 2, "username": "theanh", "email": "theanh@gmail.com", "first_name": "The", "last_name": "Anh", "phone_number": "0987654321", "address": "Hanoi"}
```

201 Created 17 ms - 473 B Save Response

Postbot Runner Start Proxy Cookies Vault Trash 3:39 PM 2/04/2025

My Workspace

assignment4 / users get user by id

GET http://localhost:8001/api/users/2/

Params Authorization Headers Body Scripts Settings

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (10) Test Results

{ JSON } Preview Visualize

```
1 [
2   "id": 2,
3   "username": "ducanh",
4   "email": "ducanh@gmail.com",
5   "first_name": "The",
6   "last_name": "Anh",
7   "phone_number": "0987654323",
8   "address": "Hanoi"
9 ]
```

200 OK 5 ms 482 B Save Response

Postbot Runner Start Proxy Cookies Vault Trash 3:40 PM 2/04/2025

My Workspace

assignment4 / users update user

PUT http://localhost:8001/api/users/1/

Params Authorization Headers (9) Body Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 [
2   "username": "ducanh2111",
3   "email": "ducanh@gmail.com",
4   "first_name": "Duc",
5   "last_name": "Anh",
6   "phone_number": "0987654321",
7   "address": "Hanoi"
8 ]
```

Body Cookies Headers (10) Test Results

{ JSON } Preview Visualize

```
1 [
2   "id": 1,
3   "username": "ducanh2111",
4   "email": "ducanh@gmail.com",
5   "first_name": "Duc",
6   "last_name": "Anh",
7   "phone_number": "0987654321",
8   "address": "Hanoi"
9 ]
```

200 OK 15 ms 486 B Save Response

Postbot Runner Start Proxy Cookies Vault Trash 3:40 PM 2/04/2025

The screenshot shows the Postman interface with a collection named "assignment4". The "products" folder contains a "DELETE" request for "http://localhost:8001/api/users/2". The response body is empty, indicating a successful delete operation.

Product:

The screenshot shows the Postman interface with a collection named "assignment4". The "products" folder contains a "GET" request for "http://localhost:8002/api/products/". The response body is a JSON array containing two product objects:

```

[{"id": 1, "name": "Laptop Lenovo", "description": "Laptop Lenovo", "price": "2888.00", "stock": 10}, {"id": 2, "name": "Laptop Dell", "description": "Laptop Dell", "price": "1888.00", "stock": 10}
]

```

The screenshot shows the Postman application interface. The left sidebar displays a collection named "assignment4" containing various API endpoints for products, users, orders, payments, and deliveries. The main workspace shows a POST request to the "/products" endpoint at `http://localhost:8003/api/products/`. The request body is set to raw JSON:

```
1 {  
2   "name": "Laptop HP",  
3   "description": "Laptop HP",  
4   "price": 1500,  
5   "stock": 50  
6 }
```

The response status is 201 Created, with a duration of 15 ms and a size of 416 B. Below the request, the response body is also shown in JSON format:

```
1 {  
2   "id": 1,  
3   "name": "Laptop HP",  
4   "description": "Laptop HP",  
5   "price": "1500.00",  
6   "stock": 10  
7 }
```

The bottom navigation bar includes links for Chirp, Find and replace, and Console, along with system status indicators for Online, Runner, Start Proxy, Cookies, Vault, and Trash.

The screenshot shows the Postman application interface. The left sidebar displays a tree view of the 'My Workspace' collection, which contains several sub-collections and their associated endpoints:

- assignment4**
 - products**
 - `POST add product`
 - `GET get all products`
 - `GET get product by id`
 - `PUT update product`
 - `DELETE delete product`
 - users**
 - `POST add user`
 - `GET get all users`
 - `GET get user by id`
 - `PUT update user`
 - `DELETE delete user`
 - orders**
 - `POST create order`
 - `POST create order detail`
 - `GET get all orders`
 - `GET get all order details`
 - `GET get order by id`
 - payments**
 - `POST create payment`
 - `PUT update payment`
 - `GET get all payments`
 - `GET get payment by id`
 - `DELETE delete payment`
 - deliveries**
 - `POST create delivery`
 - `PUT update delivery`
 - `GET get all deliveries`
 - `GET get delivery by id`
 - `DELETE delete delivery`

The screenshot shows the Postman interface with a collection named "assignment4". A "products" endpoint is selected. A PUT request is being made to `http://localhost:8002/api/products/3/`. The "Body" tab is selected, showing a JSON payload:

```

1 [
2   {
3     "id": 3,
4     "name": "Laptop HP",
5     "description": "Laptop HP",
6     "price": 2800,
7     "stock": 10
8   }
9 ]

```

The response status is 200 OK, with a duration of 8 ms and a size of 425 B. The response body is identical to the request body.

The screenshot shows the Postman interface with the same "assignment4" collection. A "products" endpoint is selected. A DELETE request is being made to `http://localhost:8002/api/products/3/`. The "Params" tab is selected, showing a "Key" parameter with the value "Key".

The response status is 204 No Content, with a duration of 13 ms and a size of 318 B. The response body is empty.

Order:

My Workspace

assignment4 / orders / create order

POST http://localhost:8003/api/orders/

Params Authorization Headers (7) Body Scripts Settings

Body ({} JSON) Preview Visualize

```

1 [
2   "user_id": 1,
3   "total_price": "200.00",
4   "status": "Pending",
5   "created_at": "2025-04-05T12:00:00"
6 ]
7

```

201 Created 2.07 s - 578 B Save Response

Body Cookies Headers (10) Test Results

Body ({} JSON) Preview Visualize

```

1 [
2   {
3     "id": 1,
4     "username": "ducanh2111",
5     "email": "ducanh@gmail.com",
6     "first_name": "Duc",
7     "last_name": "Anh",
8     "phone_number": "0987654321",
9     "address": "Hanoi"
10   }
11 ]
12

```

Postbot Runner Start Proxy Cookies Vault Trash

3:42 PM 2/04/2025

My Workspace

assignment4 / orders / get all orders

GET http://localhost:8003/api/orders/

Params Authorization Headers (7) Body Scripts Settings

Query Params

Key	Value	Description
		Description

Body Cookies Headers (10) Test Results

Body ({} JSON) Preview Visualize

```

1 [
2   {
3     "id": 1,
4     "total_price": "100.00",
5     "status": "Pending",
6     "created_at": "2025-04-05T16:02:37.734643Z",
7     "user": {
8       "id": 1,
9       "username": "ducanh2111",
10      "email": "ducanh@gmail.com",
11      "first_name": "Duc",
12      "last_name": "Anh",
13      "phone_number": "0987654321",
14      "address": "Hanoi"
15     }
16   },
17   {
18     "id": 2,
19     "total_price": "200.00",
20     "status": "Pending",
21     "created_at": "2025-04-05T16:00:58.643770Z",
22     "user": {

```

200 OK 8.20 s - 1.28 KB Save Response

Postbot Runner Start Proxy Cookies Vault Trash

3:42 PM 2/04/2025

The screenshot shows the Postman application interface. The left sidebar displays a collection named "assignment4" containing various API endpoints for products, users, orders, payments, and deliveries. The main workspace shows a POST request to "http://localhost:8003/api/orders/order-details/" with a raw JSON body:

```
POST http://localhost:8003/api/orders/order-details/
{
  "product": 4,
  "order_id": 2,
  "quantity": 1,
  "price": 59.99
}
```

The response tab shows a successful 201 Created status with the following JSON data:

```
201 Created - 17 ms - 396 B
{
  "id": 2,
  "order": 4,
  "product_id": 2,
  "quantity": 1,
  "price": "59.99"
}
```

At the bottom, the footer includes links for Chirp, Find and replace, Console, Postbot, Runner, Start Proxy, Cookies, Vault, and Trash, along with system status icons for Online, Find and replace, Console, Postbot, Runner, Start Proxy, Cookies, Vault, and Trash.

The screenshot shows the Postman application interface. The left sidebar displays 'My Workspace' with various collections like 'assignment4' and 'products'. The main area shows a request for 'GET /orders/get-all-order-details'. The response body is a JSON array of order items:

```
1 [  
2   {  
3     "id": 1,  
4     "order": 1,  
5     "product_id": 1,  
6     "quantity": 2,  
7     "price": "60.00",  
8     "product": [  
9       {  
10         "id": 1,  
11         "name": "Laptop Lenovo",  
12         "description": "Laptop Lenovo",  
13         "price": "2000.00",  
14         "stock": 10  
15       },  
16     ],  
17   },  
18   {  
19     "id": 2,  
20     "order": 4,  
21     "product_id": 2,  
22     "quantity": 1,  
23     "price": "50.00",  
24     "product": [  
25       {  
26         "id": 2,  
27         "name": "Laptop Dell",  
28         "description": "Laptop Dell",  
29         "price": "1000.00",  
30         "stock": 10  
31     }  
32   ]  
33 ]
```

My Workspace

assignment4

products

orders

payments

deliveries

Body

```
{
  "id": 1,
  "order": 1,
  "product_id": 1,
  "quantity": 2,
  "price": "59.00",
  "product": {
    "name": "Laptop Lenovo",
    "description": "Laptop Lenovo",
    "price": "2889.00",
    "stock": 10
  }
}
```

200 OK 2.05 s - 507 B

Payment:

My Workspace

assignment4

products

orders

payments

deliveries

Body

```
{
  "order_id": 4,
  "payment_status": "Pending",
  "payment_method": "Credit Card",
  "transaction_id": "123456789"
}
```

201 Created 20 ms - 443 B

My Workspace

assignment4 / payments / update payment

PUT http://localhost:8004/api/payments/2/

Body: JSON

```
[{"id": 4, "order_id": 4, "payment_status": "Completed", "payment_method": "Credit Card", "transaction_id": "123456789F"}]
```

Response: 200 OK

assignment4 / payments / get all payments

GET http://localhost:8004/api/payments/

Body: JSON

```
[{"id": 2, "order_id": 4, "payment_status": "Completed", "payment_method": "Credit Card", "transaction_id": "123456789F"}, {"id": null, "order": null}]
```

My Workspace

assignment4 / payments / get all payments

GET http://localhost:8004/api/payments/

Body: JSON

```
[{"id": 2, "order_id": 4, "payment_status": "Completed", "payment_method": "Credit Card", "transaction_id": "123456789F"}, {"id": null, "order": null}]
```

The screenshot shows the Postman interface with a collection named "assignment4". A GET request is made to `http://localhost:8004/api/payments/`. The response status is 200 OK, with a response time of 2.05 s and a size of 455 B. The response body is a JSON object:

```
{
  "id": 2,
  "order_id": 4,
  "payment_status": "Completed",
  "payment_method": "Credit Card",
  "transaction_id": "123456DEF",
  "order": null
}
```

The screenshot shows the Postman interface with the same collection "assignment4". A DELETE request is made to `http://localhost:8004/api/payments/2/`. The response status is 204 No Content, with a response time of 7 ms and a size of 318 B.

Delivery:

The screenshot shows the Postman application interface. The left sidebar displays a tree view of collections and environments. The main workspace is titled "assignment4 / deliveries / create delivery". A POST request is being prepared to the URL `http://localhost:8005/spid/deliveries/`. The "Body" tab is selected, showing the following JSON payload:

```
1 {  
2   "order_id": 1,  
3   "delivery_status": "Pending",  
4   "tracking_number": "TRACK123456",  
5   "estimated_delivery_date": "2024-10-01"  
6 }
```

The "Test Results" tab shows a successful response with status code 201 Created, time 27 ms, and size 455 B. The bottom navigation bar includes links for ChinNet, Find and replace, Console, Postbot, Runner, Start Proxy, Cookies, Vault, and Trash.

The screenshot shows the Postman application interface. The left sidebar displays a tree view of 'My Workspace' collections, including 'assignment4' which contains 'products', 'users', 'orders', and 'deliveries'. The 'deliveries' collection is currently selected, and its sub-items are listed: 'POST create delivery', 'PUT update delivery', 'GET get all deliveries', 'GET get delivery by id', and 'DELETE delete delivery'. The main workspace shows a 'PUT update delivery' request. The 'Body' tab is selected, displaying the following JSON payload:

```
1 {  
2   "order_id": 1,  
3   "delivery_status": "Shipped",  
4   "tracking_number": "TRACK123456",  
5   "estimated_delivery_date": "2024-10-02"  
6 }
```

The 'Headers' tab shows the URL as `http://localhost:8005/api/deliveries/1/`. The 'Test Results' tab shows a successful response with status `200 OK`, `15 ms`, and `464 B`. The bottom navigation bar includes links for 'Postbot', 'Runner', 'Start Proxy', 'Cookies', 'Vault', 'Trash', 'Find and replace', and 'Console'.

My Workspace

assignment4 / deliveries / get all deliveries

GET http://localhost:8005/api/deliveries/

Params Authorization Headers (7) Body Scripts Settings

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (10) Test Results

{ JSON Preview Visualize }

```
1 [  
2   {  
3     "id": 1,  
4     "order_id": 1,  
5     "delivery_status": "Shipped",  
6     "tracking_number": "TRACK123456",  
7     "estimated_delivery_date": "2024-10-02",  
8     "order":  
9       {  
10         "id": 1,  
11         "customer": 1,  
12         "quantity": 2,  
13         "price": "50.00",  
14         "product": {  
15           "id": 1,  
16           "name": "laptop Lenovo",  
17           "description": "laptop Lenovo",  
18           "price": "2000.00",  
19           "stock": 10  
20         }  
21       }  
22   }  
23 ]
```

200 OK 4.11 s - 609 B Save Response

Postbot Runner Start Proxy Cookies Vault Trash

3:52 PM 2/04/2025

My Workspace

assignment4 / deliveries / get delivery by id

GET http://localhost:8005/api/deliveries/1/

Params Authorization Headers (7) Body Scripts Settings

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (10) Test Results

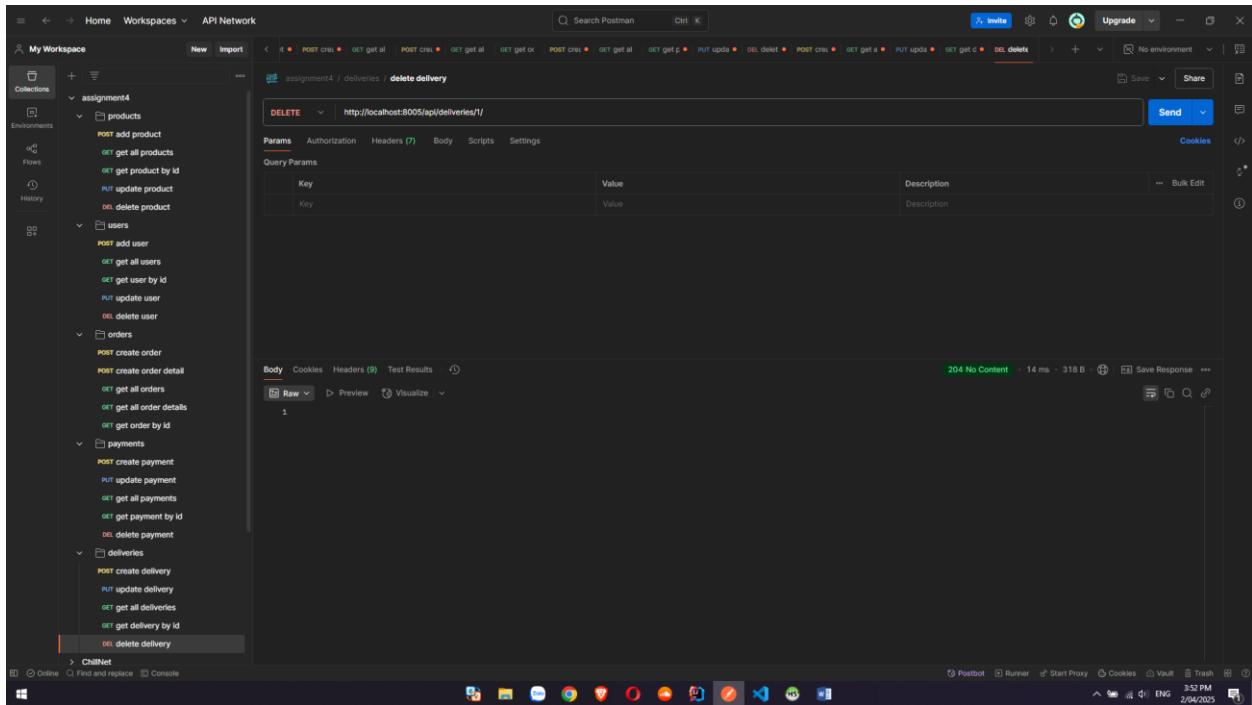
{ JSON Preview Visualize }

```
1 {  
2   "id": 1,  
3   "order_id": 1,  
4   "delivery_status": "Shipped",  
5   "tracking_number": "TRACK123456",  
6   "estimated_delivery_date": "2024-10-02"  
7 }
```

200 OK 6 ms - 464 B Save Response

Postbot Runner Start Proxy Cookies Vault Trash

3:52 PM 2/04/2025



2.4 Conclusion

Tổng kết việc triển khai hệ thống microservices

Sau quá trình thiết kế và triển khai năm service chính trong hệ thống, em đã xây dựng được một kiến trúc microservices hoàn chỉnh với những đặc điểm nổi bật như phân tán, tính linh hoạt cao, và dễ dàng mở rộng quy mô. Mỗi service được thiết kế để hoạt động độc lập, thực hiện một chức năng cụ thể và chuyên biệt, nhưng đồng thời vẫn có khả năng liên kết chặt chẽ với các service khác thông qua các giao thức RESTful API.

Các lợi ích nổi bật mà hệ thống đạt được

- 1. Tính tách biệt cao – dễ bảo trì và phát triển:
Kiến trúc microservices cho phép mỗi thành phần hoạt động như một ứng dụng riêng biệt. Điều này giúp:
 - Dễ dàng xác định và xử lý lỗi tại từng service mà không ảnh hưởng đến toàn bộ hệ thống.
 - Triển khai các bản cập nhật hoặc thêm tính năng mới nhanh chóng và linh hoạt.
 - Cho phép các team làm việc song song trên các module khác nhau.

- 2. **Khả năng mở rộng độc lập:**
Mỗi service có thể được scale riêng biệt dựa trên khối lượng truy cập hoặc nhu cầu xử lý. Ví dụ:
 - Product Service có thể mở rộng theo chiều ngang nếu lượng truy vấn sản phẩm tăng cao.
 - Cart Service có thể duy trì ở mức tài nguyên thấp hơn nếu không quá tải, giúp tiết kiệm tài nguyên hệ thống.
- 3. **Tích hợp hiệu quả với hệ thống bên ngoài:**
Hệ thống hỗ trợ tích hợp với các dịch vụ của bên thứ ba như:
 - Cổng thanh toán (Payment Gateway) để xử lý giao dịch.
 - Dịch vụ vận chuyển (Delivery API) để theo dõi tình trạng đơn hàng.
 - OAuth hoặc các hệ thống xác thực bên ngoài để hỗ trợ đăng nhập đa nền tảng.

Đặc biệt, User Service và Product Service được thiết kế để dễ dàng mở rộng chức năng hoặc tích hợp vào các hệ thống khác trong tương lai.

- 4. **Hiệu suất và khả năng phản hồi được cải thiện:**
Việc phân chia hệ thống thành nhiều service giúp:
 - Phân tán tải (load balancing) tốt hơn, tránh hiện tượng “điểm nghẽn cốt chai” (bottleneck).
 - Tăng tốc độ phản hồi cho người dùng nhờ các service xử lý song song.
 - Dễ dàng áp dụng cơ chế cache hoặc hàng đợi (message queue) tùy theo từng trường hợp cụ thể.

Một số thách thức và hướng khắc phục

Tuy đạt được nhiều lợi ích rõ rệt, nhưng mô hình microservices cũng mang lại không ít khó khăn, trong đó đáng kể nhất là:

- 1. **Đồng bộ và quản lý dữ liệu phân tán:**

- Dữ liệu được lưu trữ riêng biệt ở từng service dẫn đến việc khó đảm bảo tính nhất quán (consistency).
- Việc đồng bộ các hành động giữa các service (như Order và Payment) cần cẩn thận để tránh sai lệch.

→ Giải pháp: Áp dụng các kỹ thuật như Event-driven architecture, message broker (RabbitMQ/Kafka) hoặc saga pattern để xử lý các giao dịch phân tán.

- 2. Quản lý giao tiếp giữa các service:

- Việc giao tiếp qua HTTP REST API hoặc message queue có thể tạo độ trễ và khó kiểm soát.

→ Giải pháp: Tối ưu hóa API Gateway, sử dụng các công cụ như service discovery, circuit breaker (ví dụ: Netflix Hystrix) để tăng tính ổn định.

- 3. Giám sát và logging phức tạp hơn:

- Với nhiều service chạy độc lập, việc theo dõi và xử lý sự cố đòi hỏi hệ thống logging và monitoring mạnh.

→ Giải pháp: Tích hợp các công cụ như ELK Stack, Prometheus + Grafana, Jaeger/Zipkin để theo dõi hoạt động và truy vết lỗi (distributed tracing).

Kết luận

Việc thiết kế và triển khai thành công 5 service chính là một bước tiến quan trọng, đặt nền móng vững chắc cho hệ thống. Không chỉ đảm bảo tính linh hoạt, hiệu quả và dễ mở rộng, hệ thống còn sẵn sàng tích hợp các công nghệ mới và mở rộng chức năng trong tương lai. Với mô hình kiến trúc này, hệ thống có đủ năng lực để phục vụ số lượng lớn người dùng và thích nghi tốt với sự thay đổi nhanh chóng của công nghệ và nhu cầu thị trường.

CHAPTER 3: AI IN E-COMMERCE

3.1. Application of Deep learning in e-commerce

Trong kỷ nguyên số hiện nay, thương mại điện tử đã trở thành một phần không thể thiếu của nền kinh tế toàn cầu. Với sự bùng nổ của dữ liệu lớn (Big Data), các doanh nghiệp đang đổi mới với cả cơ hội và thách thức trong việc khai thác hiệu quả nguồn dữ liệu khổng lồ từ hành vi người dùng, giao dịch, đánh giá sản phẩm, và tương tác trên nền tảng. **Deep Learning (Học sâu)** đã nổi lên như một công nghệ đột phá, giúp biến những dữ liệu thô thành các insight có giá trị, từ đó tối ưu hóa trải nghiệm người dùng, nâng cao hiệu quả kinh doanh và vận hành.

1. Nhận diện hình ảnh sản phẩm (Product Image Recognition)

Công nghệ áp dụng:

- **CNN (Convolutional Neural Networks)**: Mô hình học sâu chuyên dụng cho xử lý hình ảnh, với khả năng tự động trích xuất đặc trưng từ pixel.
- **Transfer Learning**: Tận dụng các mô hình pre-trained như ResNet, EfficientNet, hoặc Vision Transformer (ViT) để giảm thời gian huấn luyện.
- **Object Detection**: YOLO (You Only Look Once) hoặc Faster R-CNN để phát hiện đa đối tượng trong ảnh.

Ứng dụng chi tiết:

- **Tự động gắn thẻ sản phẩm**: Hệ thống có thể phân loại sản phẩm dựa trên hình ảnh (ví dụ: giày thể thao, váy dạ hội) mà không cần nhập liệu thủ công.
- **Tìm kiếm bằng hình ảnh (Visual Search)**: Người dùng tải lên ảnh sản phẩm, hệ thống sẽ trả về các sản phẩm tương tự dựa trên đặc trưng hình ảnh. Ví dụ: Pinterest Lens hay Google Shopping.
- **Kiểm duyệt nội dung**: Phát hiện ảnh vi phạm (hàng giả, nội dung nhạy cảm) bằng cách so sánh với cơ sở dữ liệu hình ảnh đã được gán nhãn.

Ví dụ thực tế:

- **Amazon StyleSnap**: Cho phép người dùng chụp ảnh trang phục và tìm kiếm sản phẩm tương tự trên nền tảng.
- **Alibaba's Image Search**: Tích hợp AI để nhận diện sản phẩm từ ảnh chụp màn hình hoặc ảnh tải lên.

2. Dự đoán hành vi người dùng (User Behavior Prediction)

Công nghệ áp dụng:

- **RNN (Recurrent Neural Networks)**: Xử lý dữ liệu chuỗi thời gian (lịch sử duyệt web, mua hàng).
- **Transformer**: Mô hình attention-based (ví dụ: BERT) để phân tích tương tác đa kênh (click, scroll, thời gian xem).

- **Gradient Boosting Trees (kết hợp Deep Learning):** Dự đoán khả năng chuyển đổi (conversion rate).

Ứng dụng chi tiết:

- **Phân khúc khách hàng:** Nhóm người dùng dựa trên hành vi (ví dụ: khách hàng tiềm năng, khách hàng trung thành).
- **Dự đoán tỷ lệ rời bỏ (Churn Prediction):** Đánh giá nguy cơ khách hàng ngừng sử dụng dịch vụ.
- **Tối ưu thời điểm tiếp thị:** Gửi thông báo khuyến mãi khi người dùng có khả năng mua cao nhất (ví dụ: cuối tuần hoặc sau giờ làm việc).

Ví dụ thực tế:

- **Netflix:** Dự đoán thời điểm người dùng có khả năng xem phim cao nhất để đề xuất nội dung.
- **Shopee:** Gửi push notification dựa trên lịch sử mua sắm và thời gian hoạt động.

3. Hệ thống đề xuất sản phẩm (Product Recommendation Systems)

Công nghệ áp dụng:

- **Collaborative Filtering + Deep Learning:** Kết hợp ma trận người dùng-sản phẩm với Autoencoder.
- **Hybrid Models:** Kết hợp dữ liệu hành vi và nội dung sản phẩm (metadata, hình ảnh).
- **Reinforcement Learning:** Tối ưu đề xuất theo thời gian thực (ví dụ: Amazon Personalize).

Ứng dụng chi tiết:

- **Đề xuất đa kênh:** Gợi ý sản phẩm trên trang chủ, email marketing, hoặc ứng dụng di động.
- **Cross-selling/Up-selling:** Đề xuất sản phẩm bổ sung (ví dụ: pin sạc dự phòng khi mua điện thoại).
- **Phiên bản cá nhân hóa:** Mỗi người dùng nhìn thấy một danh sách sản phẩm khác nhau.

Ví dụ thực tế:

- **Amazon's "Frequently Bought Together":** Sử dụng mô hình đồng xuất hiện (co-occurrence) và học sâu.
- **TikTok Shop:** Đề xuất sản phẩm dựa trên video người dùng xem.

4. Xử lý ngôn ngữ tự nhiên (NLP)

Công nghệ áp dụng:

- **Sentiment Analysis:** BERT hoặc RoBERTa để phân tích cảm xúc đánh giá.
- **Chatbot:** GPT-3 hoặc DialoGPT để trả lời tự động câu hỏi thường gặp.
- **Text Generation:** Tự động tạo mô tả sản phẩm từ thông số kỹ thuật.

Ứng dụng chi tiết:

- **Tóm tắt đánh giá:** Trích xuất điểm chính từ review dài.
- **Phát hiện review giả:** Dựa trên mẫu ngôn ngữ bất thường.
- **Hỗ trợ đa ngôn ngữ:** Dịch tự động mô tả sản phẩm sang ngôn ngữ địa phương.

5. Phát hiện gian lận (Fraud Detection)

Công nghệ:

- **Autoencoders:** Phát hiện giao dịch bất thường bằng cách tái tạo dữ liệu.
- **Graph Neural Networks (GNN):** Phân tích mạng lưới tài khoản gian lận.

Ứng dụng:

- **Phát hiện thẻ tín dụng giả:** So sánh với mẫu giao dịch hợp lệ.
- **Chống click fraud:** Ngăn chặn lượt click quảng cáo giả mạo.

6. Quản lý kho và chuỗi cung ứng

Công nghệ:

- **LSTM/Prophet:** Dự báo nhu cầu theo mùa.
- **Computer Vision:** Kiểm kho tự động bằng drone hoặc camera.

Lợi ích:

- Giảm 30% chi phí tồn kho.
- Tối ưu lộ trình giao hàng bằng phân tích GPS và lịch sử giao nhận.

Kết luận

Deep Learning đang định hình lại thương mại điện tử thông qua:

1. **Cá nhân hóa:** Trải nghiệm "một-đến-một" cho từng khách hàng.
2. **Tự động hóa:** Giảm sự can thiệp thủ công trong quản lý và vận hành.

- Tối ưu chi phí:** Giảm lãng phí tài nguyên nhờ dự báo chính xác.
- Bảo mật:** Nâng cao an toàn giao dịch và dữ liệu người dùng.

Các doanh nghiệp áp dụng Deep Learning sớm sẽ có lợi thế cạnh tranh vượt trội trong thị trường đầy biến động.

3.2. Sentiment analysis

1. Khái Niệm Tổng Quan

Phân tích cảm xúc (Sentiment Analysis) là một lĩnh vực quan trọng trong xử lý ngôn ngữ tự nhiên (NLP), ứng dụng các kỹ thuật học máy (Machine Learning) và học sâu (Deep Learning) để xác định thái độ, cảm xúc và quan điểm của người dùng thông qua các đoạn văn bản. Các nguồn dữ liệu phổ biến bao gồm đánh giá sản phẩm, bình luận trên mạng xã hội, phản hồi dịch vụ khách hàng, bài viết blog, tin tức và các cuộc khảo sát.

Trong bối cảnh thương mại điện tử, Sentiment Analysis giúp doanh nghiệp hiểu rõ tâm lý khách hàng, từ đó tối ưu hóa chiến lược kinh doanh, nâng cao chất lượng sản phẩm/dịch vụ và cải thiện trải nghiệm người dùng. Nhờ sự phát triển của AI, đặc biệt là các mô hình học sâu như BERT, GPT và Transformer, việc phân tích cảm xúc từ hàng triệu đánh giá trở nên chính xác và hiệu quả hơn bao giờ hết.

2. Ứng Dụng Cụ Thể trong Thương Mại Điện Tử

2.1. Đánh Giá Mức Độ Hài Lòng của Khách Hàng

- Tự động phân loại cảm xúc:** Các thuật toán NLP sẽ phân tích văn bản và phân loại thành 3 nhóm chính:
 - Tích cực (Positive):** Đánh giá tốt, khen ngợi sản phẩm.
 - Trung lập (Neutral):** Thông tin khách quan, không bày tỏ rõ cảm xúc.
 - Tiêu cực (Negative):** Phàn nàn, chỉ trích hoặc yêu cầu hoàn tiền.
- Giám sát và cảnh báo tự động:** Hệ thống có thể gửi thông báo cho bộ phận chăm sóc khách hàng khi phát hiện đánh giá tiêu cực, giúp xử lý nhanh chóng và giảm thiểu rủi ro về uy tín thương hiệu.

2.2. Hỗ Trợ Hệ Thống Đề Xuất Thông Minh

- Ưu tiên sản phẩm được đánh giá cao:** Các mặt hàng có nhiều phản hồi tích cực sẽ được hiển thị nổi bật hơn trong kết quả tìm kiếm hoặc trang chủ.
- Lọc sản phẩm kém chất lượng:** Những sản phẩm nhận quá nhiều đánh giá tiêu cực có thể bị gỡ bỏ hoặc đưa vào danh sách cảnh báo.

- **Gợi ý sản phẩm tương tự:** Nếu một sản phẩm bị đánh giá thấp, hệ thống có thể đề xuất các lựa chọn thay thế tốt hơn dựa trên phân tích cảm xúc.

2.3. Phân Tích Xu Hướng Thị Trường

- **Theo dõi biến động cảm xúc theo thời gian:** Doanh nghiệp có thể nhận diện xu hướng tiêu dùng, ví dụ:
 - Sản phẩm nào đang được yêu thích?
 - Dịch vụ nào đang bị phàn nàn nhiều?
- **Dự đoán nhu cầu khách hàng:** Phân tích từ khóa và ngữ cảnh giúp doanh nghiệp điều chỉnh chiến lược tiếp thị, ví dụ:
 - Tăng quảng cáo sản phẩm đang "hot".
 - Cải tiến hoặc ngừng kinh doanh sản phẩm có phản hồi kém.

2.4. Tăng Độ Tin Cậy và Tỷ Lệ Chuyển Đổi

- **Hiển thị đánh giá thực tế:** Khách hàng mới thường dựa vào phản hồi của người dùng trước đó để ra quyết định mua hàng.
- **Giảm tỷ lệ hủy đơn hàng:** Khi sản phẩm có nhiều đánh giá chân thực, khách hàng sẽ tin tưởng hơn, từ đó giảm tỷ lệ trả hàng hoặc khiếu nại.

2.5. Phát Hiện Đánh Giá Gian Lận và Spam

- **Lọc bình luận fake (mua like, đánh giá tiêu cực cạnh tranh):** Sử dụng AI để phát hiện các đánh giá không tự nhiên, có dấu hiệu spam hoặc được tạo ra từ tài khoản ảo.
- **Xác định "khách hàng khó tính":** Nhận diện những người dùng có xu hướng đánh giá tiêu cực để có chính sách hỗ trợ phù hợp.

3. Quy Trình Thực Hiện Sentiment Analysis

3.1. Thu Thập Dữ Liệu

- **Nguồn dữ liệu đa dạng:**
 - Đánh giá trên các sàn thương mại điện tử (Shopee, Lazada, Tiki).
 - Bình luận Facebook, YouTube, TikTok, Twitter.
 - Phản hồi qua email, chatbot, hoặc cuộc gọi CSKH.
- **Dữ liệu có cấu trúc và phi cấu trúc:** Có thể là văn bản ngắn (1-2 câu) hoặc dài (bài viết blog, bài đánh giá chi tiết).

3.2. Tiền Xử Lý Văn Bản (Text Preprocessing)

- **Làm sạch dữ liệu:**
 - Loại bỏ stopwords (từ không mang nghĩa như "và", "hoặc", "là").

- Chuẩn hóa chữ thường, xóa ký tự đặc biệt, dấu câu.
- Xử lý từ viết tắt, tiếng lóng (vd: "khá ổn" → "tốt", "tệ" → "xấu").
- **Tách từ (Tokenization) và gán nhãn từ loại (POS Tagging).**

3.3. Biểu Diễn Văn Bản

- **Phương pháp truyền thống:**
 - **Bag-of-Words (BoW):** Đếm tần suất xuất hiện của từ.
 - **TF-IDF:** Đánh giá mức độ quan trọng của từ trong văn bản.
- **Phương pháp hiện đại:**
 - **Word Embedding (Word2Vec, GloVe):** Biểu diễn từ dưới dạng vector.
 - **Transformer-based (BERT, RoBERTa, GPT):** Hiểu ngữ cảnh sâu hơn nhờ cơ chế Attention.

3.4. Huấn Luyện Mô Hình

- **Mô hình học máy:**
 - SVM, Naive Bayes, Random Forest (phù hợp với dữ liệu ít).
- **Mô hình học sâu:**
 - **LSTM/GRU:** Xử lý tốt văn bản dài, có ngữ cảnh phức tạp.
 - **CNN cho văn bản:** Trích xuất đặc trưng cục bộ.
 - **BERT & Transformer:** Cho độ chính xác cao nhất hiện nay.

3.5. Đánh Giá Mô Hình

- **Các chỉ số quan trọng:**
 - **Accuracy (Độ chính xác tổng thể).**
 - **Precision (Độ chính xác khi dự đoán lớp).**
 - **Recall (Khả năng phát hiện đúng cảm xúc).**
 - **F1-Score (Cân bằng giữa Precision và Recall).**

4. Lợi Ích và Thách Thức

4.1. Lợi Ích

- **Nâng cao trải nghiệm khách hàng:** Phản hồi nhanh chóng với các đánh giá tiêu cực.
- **Tối ưu chiến lược marketing:** Điều chỉnh quảng cáo dựa trên phân tích tâm lý người dùng.
- **Giảm chi phí CSKH:** Tự động hóa quy trình phân loại và xử lý phản hồi.

4.2. Thách Thức

- **Đa nghĩa và ngôn ngữ tự nhiên:** Một câu có thể mang nhiều sắc thái cảm xúc khác nhau.
- **Tiếng lóng, phương ngữ địa phương:** Khó chuẩn hóa trong các ngôn ngữ như tiếng Việt.
- **Dữ liệu không cân bằng:** Số lượng đánh giá tích cực thường nhiều hơn tiêu cực, dẫn đến mô hình bị thiên lệch.

5. Xu Hướng Phát Triển Trong Tương Lai

- **Kết hợp Multimodal Sentiment Analysis:** Phân tích cả văn bản, hình ảnh, giọng nói để hiểu sâu hơn cảm xúc người dùng.
- **AI có khả năng hiểu ngữ cảnh phức tạp:** Các mô hình như ChatGPT, Gemini sẽ giúp phân tích cảm xúc chính xác hơn.
- **Ứng dụng trong chatbot AI:** Hỗ trợ phản hồi tự động dựa trên tâm trạng khách hàng.

Kết Luận

Phân tích cảm xúc không chỉ là công cụ hỗ trợ doanh nghiệp mà còn là yếu tố then chốt giúp thương mại điện tử phát triển bền vững. Với sự tiến bộ của AI, các giải pháp Sentiment Analysis ngày càng thông minh, mang lại lợi ích thiết thực cho cả người mua lẫn người bán. Trong tương lai, công nghệ này sẽ tiếp tục được cải tiến, trở thành một phần không thể thiếu trong hành trình chuyển đổi số của các doanh nghiệp.

3.3. Applying sentiment analysis for recommendation

1. Giới Thiệu Tổng Quan

Trong thời đại bùng nổ thương mại điện tử, **hệ thống gợi ý (Recommendation System)** đóng vai trò quan trọng trong việc thúc đẩy doanh số và tăng tỷ lệ chuyển đổi. Tuy nhiên, các phương pháp truyền thống như **Collaborative Filtering (CF)** hay **Content-Based Filtering** thường chỉ dựa trên dữ liệu hành vi (lượt xem, mua hàng, đánh giá sao) mà bỏ qua yếu tố quan trọng: **cảm xúc thực sự của người dùng** ẩn chứa trong các đánh giá văn bản.

Phân tích cảm xúc (Sentiment Analysis) ra đời như một giải pháp tối ưu, giúp hệ thống hiểu sâu hơn về **thái độ, sở thích và mức độ hài lòng** của khách hàng thông qua ngôn ngữ tự nhiên. Bằng cách kết hợp AI, NLP và Deep Learning, doanh nghiệp có thể xây dựng hệ thống gợi ý thông minh hơn, cá nhân hóa hơn và giảm thiểu các đề xuất không phù hợp.

2. Tại Sao Cần Kết Hợp Sentiment Analysis Vào Hệ Thống Gợi Ý?

2.1. Hạn Chế Của Hệ Thống Gợi Ý Truyền Thống

- **Chỉ dựa vào điểm số (rating):**
 - Một đánh giá 4 sao có thể chứa nội dung tiêu cực (ví dụ: "Sản phẩm tốt nhưng giao hàng chậm").
 - Người dùng có xu hướng đánh giá cao ngay cả khi không hài lòng (hiệu ứng "không muốn làm mất lòng người bán").
- **Bỏ qua ngữ cảnh cảm xúc:**
 - Cùng một sản phẩm, nhưng người dùng khác nhau có trải nghiệm cảm xúc khác nhau (ví dụ: áo thun "co giãn tốt" với người thích ôm body vs. người thích rộng rãi).

2.2. Lợi Thế Khi Tích Hợp Sentiment Analysis

Hiểu rõ hơn về nhu cầu thực sự của người dùng thông qua ngôn ngữ tự nhiên.

Loại bỏ đề xuất gây khó chịu (ví dụ: gợi ý giày thể thao cho người đã phàn nàn về đế cứng).

Tăng độ chính xác của gợi ý bằng cách kết hợp **cảm xúc + hành vi**.

Phát hiện xu hướng mới từ phân tích cảm xúc theo nhóm người dùng (ví dụ: Gen Z thích phong cách streetwear, trong khi dân văn phòng ưa tối giản).

3. Cách Tích Hợp Sentiment Analysis Vào Hệ Thống Gọi Ý

3.1. Quy Trình Chi Tiết

Bước 1: Thu Thập Dữ Liệu Đánh Giá

- **Nguồn dữ liệu:**
 - Đánh giá sản phẩm trên website (text + rating).
 - Bình luận mạng xã hội (Facebook, Tiktok).
 - Phản hồi qua email, chatbot.
- **Đặc điểm dữ liệu:**
 - Ngắn (1-2 câu) hoặc dài (bài đánh giá chi tiết).
 - Có thể chứa tiếng lóng, biểu cảm (emoji), ngôn ngữ địa phương.

Bước 2: Tiền Xử Lý Văn Bản

- **Làm sạch dữ liệu:**
 - Xóa stopwords ("à", "ở", "và"), ký tự đặc biệt, HTML tags.
 - Chuẩn hóa từ viết tắt ("ship" → "giao hàng", "xịn" → "tốt").
 - Xử lý emoji
- **Phân tích ngữ cảnh:**
 - Nhận diện câu mỉa mai ("Đóng gói tuyệt vời, hàng đến vỡ tan tành").

Bước 3: Phân Loại Cảm Xúc

- **Phương pháp:**
 - **Rule-Based:** Dùng từ điển cảm xúc (ví dụ: "tuyệt vời" → +1, "tê" → -1).
 - **Machine Learning:** SVM, Naive Bayes (phù hợp dữ liệu nhỏ).

- **Deep Learning:**
 - **LSTM/GRU:** Phân tích chuỗi văn bản dài.
 - **BERT (Transformers):** Hiểu ngữ nghĩa sâu, cho kết quả chính xác nhất.
- **Đầu ra:**
 - **Nhận cảm xúc:** Positive (Tích cực), Neutral (Trung lập), Negative (Tiêu cực).
 - **Điểm số:** Từ -1 (rất tiêu cực) đến +1 (rất tích cực).

Bước 4: Kết Hợp Vào Hệ Thống Gọi Ý

- **Cải tiến Collaborative Filtering:**
 - Thay vì chỉ dựa vào rating, tính toán **độ tương đồng người dùng** dựa trên **cảm xúc + hành vi**.
 - Ví dụ: User A và User B cùng đánh giá 4 sao, nhưng nếu cả hai đều **khen chất liệu**, hệ thống sẽ ưu tiên gợi ý sản phẩm tương tự.
- **Content-Based Filtering nâng cao:**
 - Gợi ý sản phẩm có **đặc điểm được khen ngợi** (ví dụ: "pin trâu" → đề xuất điện thoại có pin tốt).
- **Real-time Recommendation:**
 - Khi người dùng đọc đánh giá tiêu cực về "màn hình dễ vỡ", hệ thống tự động **giảm ưu tiên** sản phẩm đó.

Bước 5: Đánh Giá & Tối Uu

- **Chỉ số đánh giá:**
 - **Click-through Rate (CTR):** Tỷ lệ người dùng click vào sản phẩm được gợi ý.
 - **Conversion Rate:** Tỷ lệ chuyển đổi thành đơn hàng.
 - **User Satisfaction Score:** Khảo sát mức độ hài lòng sau khi nhận gợi ý.

4. Ví Dụ Minh Họa

Người Dùng	Đánh Giá Sản Phẩm A	Điểm Cảm Xúc	Gợi Ý Sản Phẩm B
user123	"Chất liệu mềm, thoái mái, xứng đáng 5 sao!"	+0.9	Áo thun cùng chất liệu cotton organic
user456	"Màu đẹp nhưng form rộng quá, mặc không đẹp"	-0.5	Áo kiểu ôm body, size nhỏ hơn
user789	"Hàng bình thường, không có gì nổi bật"	0	Sản phẩm khác cùng giá nhưng có tính năng mới

5. Lợi Ích Vượt Trội

5.1. Với Doanh Nghiệp

- **Tăng doanh thu:** Gợi ý chính xác giúp khách hàng mua nhiều hơn.
- **Giảm tỷ lệ trả hàng:** Loại bỏ sản phẩm không phù hợp ngay từ khâu đàm phán.
- **Cạnh tranh tốt hơn:** Hiểu insight khách hàng để điều chỉnh chiến lược.

5.2. Với Người Dùng

- **Tiết kiệm thời gian:** Nhận gợi ý sát với nhu cầu thực tế.
- **Trải nghiệm cá nhân hóa:** Cảm thấy "được thấu hiểu" bởi hệ thống.

6. Thách Thức & Giải Pháp

6.1. Thách Thức

- **Đa nghĩa trong ngôn ngữ:**
 - Ví dụ: "Chất liệu nhẹ nhàng quá mỏng" → vừa tích cực, vừa tiêu cực.
- **Xử lý tiếng địa phương, tiếng lóng:**
 - "Chán chết đi được" (miền Bắc) vs. "Dở ẹc" (miền Nam).
- **Tính toán phức tạp:**
 - Mô hình BERT yêu cầu GPU mạnh để xử lý real-time.

6.2. Giải Pháp

- **Fine-tuning BERT trên dataset tiếng Việt** để hiểu ngữ cảnh địa phương.
- **Kết hợp nhiều phương pháp:**
 - Dùng Rule-Based để lọc câu đơn giản, Deep Learning cho câu phức tạp.
- **Ứng dụng Cloud Computing** (AWS, Google Cloud) để giảm chi phí tính toán.

7. Xu Hướng Tương Lai

- **Multimodal Recommendation:**
 - Kết hợp text + hình ảnh + video review để phân tích cảm xúc toàn diện.
- **Explainable AI (XAI):**
 - Giải thích lý do gợi ý (ví dụ: "Gợi ý này vì 90% người khen đẹp").
- **Real-time Sentiment Tracking:**
 - Theo dõi cảm xúc người dùng **ngay khi họ đang xem sản phẩm** để điều chỉnh gợi ý tức thì.

Kết Luận

Việc tích hợp **Sentiment Analysis** vào **hệ thống gợi ý** không chỉ là xu hướng công nghệ mà còn là **bước tiến quan trọng** trong việc cá nhân hóa trải nghiệm khách hàng. Với sự phát triển của **AI và Big Data**, các nền tảng thương mại điện tử có thể hiểu người dùng sâu sắc hơn bao giờ hết, từ đó tối ưu hóa chiến lược kinh doanh và xây dựng lòng trung thành thương hiệu. Trong tương lai, hệ thống gợi ý sẽ không chỉ "biết" bạn thích gì, mà còn "hiểu" tại sao bạn thích điều đó!

3.4. Deployment

a) Xây dựng mô hình học sâu để phân loại cảm xúc

import thư viện:

```
[1]: import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

from sklearn.feature_extraction.text import CountVectorizer
from tensorflow.keras.preprocessing.text import Tokenizer # Keras từ tensorflow
from tensorflow.keras.preprocessing.sequence import pad_sequences # Keras từ tensorflow
from tensorflow.keras.models import Sequential # Keras từ tensorflow
from tensorflow.keras.layers import Dense, Embedding, LSTM, SpatialDropout1D # Keras từ tensorflow
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical # Keras từ tensorflow
import re
```

Chuẩn bị dữ liệu:

```
[8]: data = pd.read_csv('Sentiment.csv')
# Keeping only the necessary columns
data = data[['text','sentiment']]

[10]: data = data[data['sentiment'] != "Neutral"]
data['text'] = data['text'].apply(lambda x: x.lower())
data['text'] = data['text'].apply((lambda x: re.sub('[^a-zA-Z0-9\s]', '',x)))

print(data[ data['sentiment'] == 'Positive'].size)
print(data[ data['sentiment'] == 'Negative'].size)
```

Chuyển văn bản thành các chuỗi số:

```
[1]: for idx, row in data.iterrows():
    row[0] = row[0].replace('rt', ' ')

max_features = 2000
tokenizer = Tokenizer(num_words=max_features, split=' ')
tokenizer.fit_on_texts(data['text'].values)
X = tokenizer.texts_to_sequences(data['text'].values)
X = pad_sequences(X)
```

Xây dựng mô hình học sâu với LSTM:

```
[12]: embed_dim = 128
lstm_out = 196

model = Sequential()
model.add(Embedding(max_features, embed_dim, input_length = X.shape[1]))
model.add(SpatialDropout1D(0.4))
model.add(LSTM(lstm_out, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(2, activation='softmax'))
model.compile(loss = 'categorical_crossentropy', optimizer='adam', metrics = ['accuracy'])
print(model.summary())

C:\Users\dell\anaconda3\Lib\site-packages\keras\src\layers\core\embedding.py:90: UserWarning: Argument `input_length` is deprecated. Just remove it.
  warnings.warn(
Model: "sequential"



| Layer (type)                         | Output Shape | Param #     |
|--------------------------------------|--------------|-------------|
| embedding (Embedding)                | ?            | 0 (unbuilt) |
| spatial_dropout1d (SpatialDropout1D) | ?            | 0           |
| lstm (LSTM)                          | ?            | 0 (unbuilt) |
| dense (Dense)                        | ?            | 0 (unbuilt) |



Total params: 0 (0.00 B)
Trainable params: 0 (0.00 B)
Non-trainable params: 0 (0.00 B)
None
```

Huấn luyện và đánh giá mô hình:

```
[14]: Y = pd.get_dummies(data['sentiment']).values
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.33, random_state = 42)
print(X_train.shape,Y_train.shape)
print(X_test.shape,Y_test.shape)

(7188, 28) (7188, 2)
(3541, 28) (3541, 2)

[16]: batch_size = 32
model.fit(X_train, Y_train, epochs = 7, batch_size=batch_size, verbose = 2)

Epoch 1/7
225/225 - 6s - 29ms/step - accuracy: 0.8143 - loss: 0.4416
Epoch 2/7
225/225 - 4s - 17ms/step - accuracy: 0.8599 - loss: 0.3300
Epoch 3/7
225/225 - 4s - 19ms/step - accuracy: 0.8798 - loss: 0.2867
Epoch 4/7
225/225 - 5s - 22ms/step - accuracy: 0.8918 - loss: 0.2648
Epoch 5/7
225/225 - 5s - 22ms/step - accuracy: 0.9011 - loss: 0.2391
Epoch 6/7
225/225 - 5s - 22ms/step - accuracy: 0.9153 - loss: 0.2095
Epoch 7/7
225/225 - 5s - 22ms/step - accuracy: 0.9253 - loss: 0.1903
[16]: <keras.src.callbacks.history.History at 0x108a838ef90>

[18]: validation_size = 1500

X_validate = X_test[-validation_size:]
Y_validate = Y_test[-validation_size:]
X_test = X_test[:-validation_size]
Y_test = Y_test[:-validation_size]
score,acc = model.evaluate(X_test, Y_test, verbose = 2, batch_size = batch_size)
print("score: %.2f" % (score))
print("acc: %.2f" % (acc))

64/64 - 1s - 13ms/step - accuracy: 0.8413 - loss: 0.4566
score: 0.46
acc: 0.84
```

Lưu mô hình và tokenizer đã được huấn luyện:

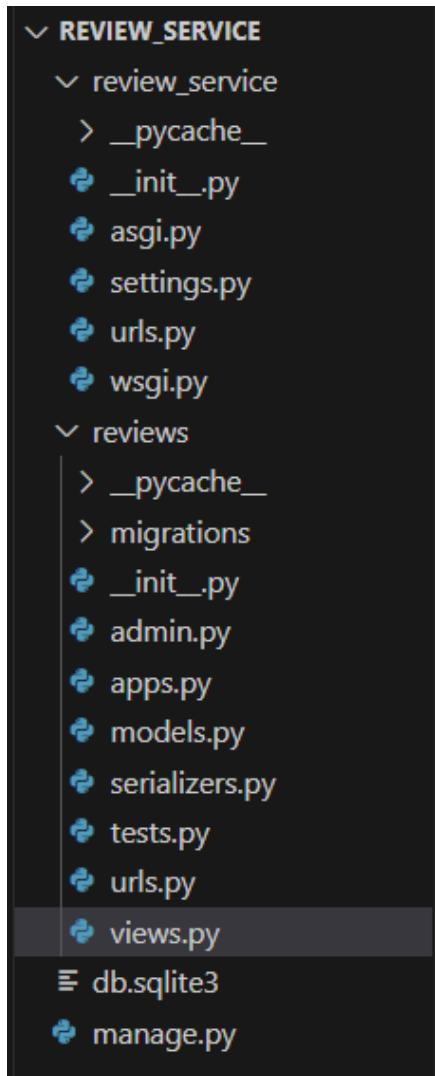
```
[24]: # Lưu mô hình vào tệp
model.save('sentiment_analysis_model.h5')

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.

[29]: import pickle

# Lưu tokenizer đã được huấn luyện
with open('tokenizer.pkl', 'wb') as handle:
    pickle.dump(tokenizer, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

b) Tạo review_service



Model Review:

```
reviews > models.py > Review
1 from django.db import models
2
3 class Review(models.Model):
4     user_id = models.IntegerField()
5     product_id = models.IntegerField()
6     rating = models.IntegerField() # 1 to 5
7     comment = models.TextField()
8     created_at = models.DateTimeField(auto_now_add=True)
9
10    def __str__(self):
11        return f"User {self.user_id} reviewed Product {self.product_id}"
12
```

Serializers:

```
reviews > serializers.py > ...
1 from rest_framework import serializers
2 from .models import Review
3
4 class ReviewSerializer(serializers.ModelSerializer):
5     class Meta:
6         model = Review
7         fields = ['id', 'user_id', 'product_id', 'rating', 'comment', 'created_at']
8
```

Views:

```
reviews > views.py > ...
1 import requests
2 from rest_framework import generics, status
3 from rest_framework.response import Response
4 from .models import Review
5 from .serializers import ReviewSerializer
6
7 USER_SERVICE_URL = 'http://localhost:8001/api/users/'
8 PRODUCT_SERVICE_URL = 'http://localhost:8002/api/products/'
9
10 class ReviewListCreateView(generics.ListCreateAPIView):
11     queryset = Review.objects.all()
12     serializer_class = ReviewSerializer
13
14     def list(self, request, *args, **kwargs):
15         reviews = self.get_queryset()
16         data = []
17
18         for review in reviews:
19             review_data = ReviewSerializer(review).data
20
```

```

20
21     # Gọi UserService
22     user_response = requests.get(f'{USER_SERVICE_URL}{review.user_id}/')
23     user_data = user_response.json() if user_response.status_code == 200 else None
24
25     # Gọi ProductService
26     product_response = requests.get(f'{PRODUCT_SERVICE_URL}{review.product_id}/')
27     product_data = product_response.json() if product_response.status_code == 200
28
29     # Thêm dữ liệu
30     review_data['user'] = user_data
31     review_data['product'] = product_data
32     data.append(review_data)
33
34 return Response(data)
35
36 def create(self, request, *args, **kwargs):
37     serializer = ReviewSerializer(data=request.data)
38     if serializer.is_valid():
39         serializer.save()
40         return Response(serializer.data, status=status.HTTP_201_CREATED)
41     return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
42
43 class ReviewRetrieveUpdateDestroyView(generics.RetrieveUpdateDestroyAPIView):
44     queryset = Review.objects.all()
45     serializer_class = ReviewSerializer
46
47     def retrieve(self, request, *args, **kwargs):
48         instance = self.get_object()
49         review_data = ReviewSerializer(instance).data
50
51         # Gọi UserService
52         user_response = requests.get(f'{USER_SERVICE_URL}{instance.user_id}/')
53         user_data = user_response.json() if user_response.status_code == 200 else None
54
55         # Gọi ProductService
56         product_response = requests.get(f'{PRODUCT_SERVICE_URL}{instance.product_id}/')
57         product_data = product_response.json() if product_response.status_code == 200 else None
58
59         review_data['user'] = user_data
60         review_data['product'] = product_data
61
62     return Response(review_data)
63
64     def update(self, request, *args, **kwargs):
65         instance = self.get_object()
66         serializer = ReviewSerializer(instance, data=request.data, partial=True)
67         if serializer.is_valid():
68             serializer.save()
69             return Response(serializer.data)
70         return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
71

```

```

71
72     def destroy(self, request, *args, **kwargs):
73         instance = self.get_object()
74         instance.delete()
75         return Response(status=status.HTTP_204_NO_CONTENT)
76

```

Urls:

```

review_service > urls.py > ...
17 from django.contrib import admin
18 from django.urls import path, include
19
20 urlpatterns = [
21     path('admin/', admin.site.urls),
22     path('api/reviews/', include('reviews.urls')),
23 ]
24

```

```

reviews > urls.py > ...
1 from django.urls import path
2 from .views import ReviewListCreateView, ReviewRetrieveUpdateDestroyView
3
4 urlpatterns = [
5     path('', ReviewListCreateView.as_view(), name='review-list-create'),
6     path('<int:pk>/', ReviewRetrieveUpdateDestroyView.as_view(), name='review-detail'),
7 ]
8

```

Test các API:

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'My Workspace' containing collections like 'assignment4' and 'reviews'. Under 'reviews', there are several items including 'POST create reviews'. The main area shows a POST request to 'http://localhost:8006/api/reviews/'. The 'Body' tab is selected, showing the following JSON payload:

```

{
    "user_id": 1,
    "product_id": 2,
    "rating": 3,
    "comment": "So disappointed"
}

```

The 'Headers' tab shows 'Content-Type: application/json'. The 'Test Results' tab indicates a '201 Created' response with a size of 452 B. The bottom status bar shows the URL 'http://localhost:8006/api/reviews/' and the time '2:53 PM 5/04/2023'.

My Workspace

assignment4

products

POST add product
GET get all products
GET get product by id
PUT update product
DELETE delete product

users

orders

payments

deliveries

reviews

POST create reviews
PUT update review
GET get all reviews
GET get review by id
DELETE delete review

sentiment

GET analyze sentiment of all reviews...
GET analyze sentiment of 1 review

ChillNet

event

library-system

microservice-django

New Collection

Project Management

ShopApp

ShopAppJavaSpringUdemy2023

ShopSheeker

Twitter

assignment4 / reviews / update review

PUT http://localhost:8006/api/reviews/1/

Params Authorization Headers Body Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 [
2   "user_id": 1,
3   "product_id": 1,
4   "rating": 4,
5   "comment": "The product is very good!",
6 ]
```

Body Cookies Headers Test Results

{ JSON } Preview Visualize

200 OK 47 ms - 471 B Save Response

Postbot Runner Start Proxy Cookies Vault Trash

Online Find and replace Console

2:53 PM 5/04/2025

My Workspace

assignment4

products

POST add product
GET get all products
GET get product by id
PUT update product
DELETE delete product

users

orders

payments

deliveries

reviews

POST create reviews
PUT update review
GET get all reviews

GET get review by id
DELETE delete review

sentiment

GET analyze sentiment of all reviews...
GET analyze sentiment of 1 review

ChillNet

event

library-system

microservice-django

New Collection

Project Management

ShopApp

ShopAppJavaSpringUdemy2023

ShopSheeker

Twitter

assignment4 / reviews / get all reviews

GET http://localhost:8006/api/reviews/

Params Authorization Headers Body Scripts Settings

Query Params

Key	Value	Description
Key		Description

Body Cookies Headers Test Results

{ JSON } Preview Visualize

200 OK - 8.19 s - 1.05 KB Save Response

Postbot Runner Start Proxy Cookies Vault Trash

Online Find and replace Console

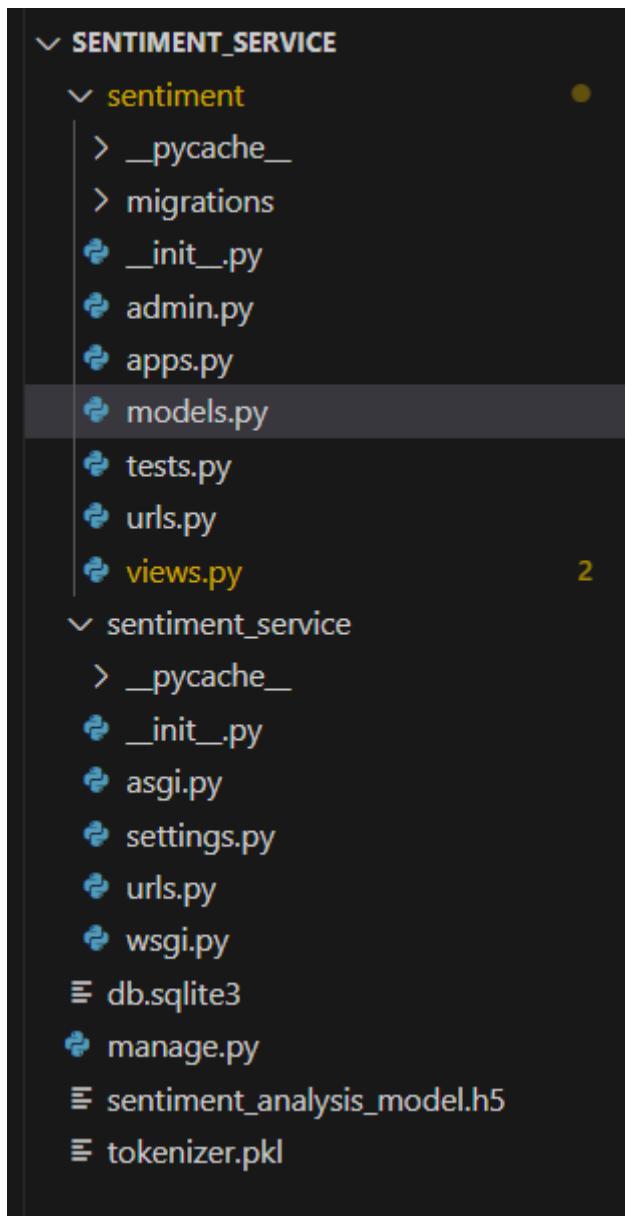
2:53 PM 5/04/2025

The screenshot shows the Postman application interface. On the left, the 'My Workspace' sidebar lists various collections and environments. The main workspace displays a collection named 'assignment4' with a 'reviews' folder. A specific API endpoint, 'get review by id', is selected. The 'Body' tab of the request panel shows a JSON payload:

```
1 {  
2   "id": 1,  
3   "user_id": 1,  
4   "product_id": 1,  
5   "rating": 4,  
6   "comment": "The product is very good!",  
7   "created_at": "2025-04-04T14:14:08.468796Z",  
8   "user": [  
9     {"id": 1,  
10     "username": "ducanh2111",  
11     "email": "ducanh@gmail.com",  
12     "first_name": "Duc",  
13     "last_name": "Anh",  
14     "phone_number": "0987654321",  
15     "address": "Hanoi"  
16   ],  
17   "product": [  
18     {"id": 1,  
19     "name": "Laptop Lenovo",  
20     "description": "Laptop Lenovo",  
21     "price": "2000.00",  
22     "stock": 10  
23   ]  
24 }
```

The response panel shows a successful '200 OK' status with a response time of 4.16 s and a size of 722 B. The response body is identical to the JSON sent in the request.

c) Tạo sentiment_service



Views:

```
sentiment > views.py > ...
1 import re
2 import numpy as np
3 import pickle
4 import requests
5 from rest_framework.views import APIView
6 from rest_framework.response import Response
7 from tensorflow.keras.models import load_model
8 from tensorflow.keras.preprocessing.sequence import pad_sequences
9
10 # Đường dẫn đến mô hình và tokenizer đã lưu
11 MODEL_PATH = 'sentiment_analysis_model.h5'
12 TOKENIZER_PATH = 'tokenizer.pkl'
13
14 # Đường dẫn API Review Service
15 REVIEW_SERVICE_URL = 'http://localhost:8006/api/reviews/'
16 |
17 # Load mô hình và tokenizer (load 1 lần khi server khởi động)
18 model = load_model(MODEL_PATH)
19 with open(TOKENIZER_PATH, 'rb') as handle:
20     tokenizer = pickle.load(handle)
21
22 # Làm sạch văn bản
23 def clean_text(text):
24     text = text.lower()
25     text = re.sub(r'[^a-zA-Z0-9\s]', '', text)
26     text = text.replace('rt', '')
27     return text
28
29 # Dự đoán cảm xúc
30 def predict_sentiment(text, maxlen=28):
31     cleaned = clean_text(text)
32     seq = tokenizer.texts_to_sequences([cleaned])
33     padded = pad_sequences(seq, maxlen=maxlen)
34     pred = model.predict(padded)[0]
35     return "Positive" if np.argmax(pred) == 1 else "Negative"
36
```

```

37 # APIView
38 class AnalyzeReviewSentiment(APIView):
39     def get(self, request):
40         try:
41             # Gọi đến review service
42             response = requests.get(REVIEW_SERVICE_URL)
43             if response.status_code != 200:
44                 return Response({"error": "Không thể lấy dữ liệu từ Review Service"})
45
46             reviews = response.json()
47             result = []
48
49             for review in reviews:
50                 comment = review.get("comment", "")
51                 sentiment = predict_sentiment(comment)
52
53                 result.append({
54                     "review_id": review["id"],
55                     "comment": comment,
56                     "sentiment": sentiment
57                 })
58
59             return Response(result)
60
61         except Exception as e:
62             return Response({"error": str(e)}, status=500)
63
64 class AnalyzeSingleReviewSentiment(APIView):
65     def get(self, request, review_id):
66         try:
67             # Gọi đến API lấy 1 review theo id
68             response = requests.get(f"{REVIEW_SERVICE_URL}{review_id}/")
69             if response.status_code != 200:
70                 return Response({"error": f"Không tìm thấy review với id = {review_id}"})
71
72             review = response.json()
73             comment = review.get("comment", "")
74             sentiment = predict_sentiment(comment)
75
76             return Response({
77                 "review_id": review["id"],
78                 "comment": comment,
79                 "sentiment": sentiment
80             })
81
82         except Exception as e:
83             return Response({"error": str(e)}, status=500)
84
85

```

Urls:

```

sentiment_service > urls.py > ...
18
19  from django.contrib import admin
20  from django.urls import path, include
21
22  urlpatterns = [
23      path('admin/', admin.site.urls),
24      path('api/sentiment/', include('sentiment.urls')),
25  ]
26

```

```

sentiment > urls.py > ...
1  from django.urls import path
2  from .views import AnalyzeReviewSentiment, AnalyzeSingleReviewSentiment
3
4  urlpatterns = [
5      path('analyze-reviews/', AnalyzeReviewSentiment.as_view(), name='analyze-reviews'),
6      path('analyze-review/<int:review_id>/', AnalyzeSingleReviewSentiment.as_view()),
7  ]
8

```

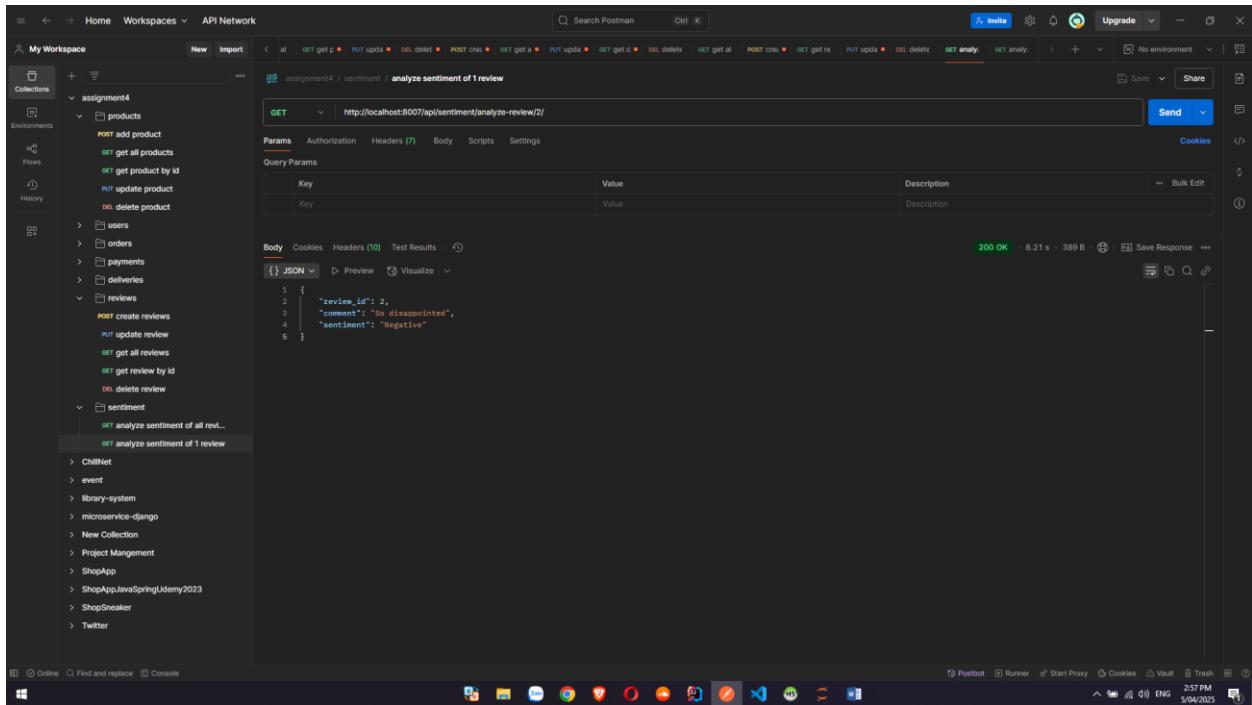
Test API:

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'My Workspace' containing collections like 'assignment4' and 'sentiment'. The main area shows a 'GET' request to 'http://localhost:8007/api/sentiment/analyze-reviews/'. The 'Body' tab is selected, showing a JSON array with two elements. Each element is an object with 'review_id', 'comment', and 'sentiment' fields.

```

1  [
2      {
3          "review_id": 1,
4          "comment": "The product is very good!",
5          "sentiment": "Positive"
6      },
7      {
8          "review_id": 2,
9          "comment": "So disappointed",
10         "sentiment": "Negative"
11     }
12 ]

```



3.5. Conclusion

Trong chương này, chúng ta đã khám phá sâu rộng những ứng dụng đột phá của **Trí tuệ nhân tạo (AI)**, đặc biệt là **Deep Learning** và **Phân tích cảm xúc (Sentiment Analysis)**, trong việc cách mạng hóa ngành thương mại điện tử. Dưới đây là tổng hợp chi tiết những đóng góp quan trọng của các công nghệ này, cùng với triển vọng phát triển trong tương lai.

1. Tổng Quan Về Ứng Dụng Deep Learning Trong Thương Mại Điện Tử

Deep Learning đã trở thành "**xương sống**" của nhiều giải pháp thông minh trong e-commerce, bao gồm:

- **Phân loại sản phẩm tự động:**
 - Sử dụng **CNN (Mạng nơ-ron tích chập)** để phân nhóm sản phẩm dựa trên hình ảnh, mô tả.
 - Ví dụ: Amazon áp dụng AI để phân loại hàng triệu sản phẩm vào danh mục chính xác chỉ trong vài giây.
- **Tìm kiếm thông minh:**

- Transformer-based models (**BERT**, **GPT**) hiểu ngữ nghĩa truy vấn người dùng, kể cả khi họ gõ sai chính tả hoặc dùng từ khóa mơ hồ.
- **Chatbot hỗ trợ khách hàng:**
 - Các chatbot AI như **Dialogflow** hay **Microsoft Bot Framework** sử dụng **NLP** để trả lời câu hỏi, xử lý đơn hàng và giải quyết khiếu nại 24/7.
- **Cá nhân hóa trải nghiệm:**
 - Hệ thống gợi ý (**Recommendation Engine**) dựa trên **LSTM** hoặc **Neural Collaborative Filtering** phân tích lịch sử mua hàng để đề xuất sản phẩm phù hợp.

→ **Kết quả:** Tăng 30% tỷ lệ chuyển đổi (theo nghiên cứu của McKinsey) và giảm 20% tỷ lệ hủy đơn do gợi ý chính xác hơn.

2. Sentiment Analysis: Công Cụ Đọc Vị Cảm Xúc Khách Hàng

Khác với đánh giá sao truyền thống, **Sentiment Analysis** khai thác **dữ liệu phi cấu trúc** (bình luận, review) để hiểu sâu hơn về trải nghiệm người dùng:

- **Phát hiện xu hướng tiêu cực ẩn giấu:**
 - Ví dụ: Một đánh giá 4 sao với nội dung "*Sản phẩm tốt nhưng đóng gói cầu thả*" sẽ được gắn nhãn "**Positive** với **điểm tiêu cực về logistics**".
- **Ứng dụng đa dạng:**
 - **Theo dõi mức độ hài lòng** theo thời gian.
 - **Lọc đánh giá gian lận** (fake review) nhờ phát hiện ngôn ngữ bất thường.
 - **Dự đoán khủng hoảng truyền thông** khi tỷ lệ bình luận tiêu cực tăng đột biến.

→ **Case study:** Shopee sử dụng **BERT tiếng Việt** để phân tích 10 triệu đánh giá mỗi ngày, giúp ưu tiên hiển thị sản phẩm có phản hồi tốt.

3. Tích Hợp Sentiment Analysis Vào Hệ Thống Gợi Ý: Bước Nhảy Vọt Về Cá Nhân Hóa

Việc kết hợp cảm xúc người dùng vào thuật toán gợi ý mang lại lợi ích vượt trội:

- **Hiểu sở thích thực sự:**

- Nếu người dùng thường xuyên khen ngợi "chất liệu thoáng khí", hệ thống sẽ ưu tiên gợi ý áo thun cotton thay vì polyester.

- **Tránh đề xuất rủi ro:**

- Sản phẩm có nhiều bình luận như "*dě rách*" sẽ bị giảm độ ưu tiên, dù rating trung bình cao.

- **Gợi ý theo ngữ cảnh cảm xúc:**

- Ví dụ: Khi người dùng đang đọc đánh giá về "laptop pin yếu", hệ thống ngay lập tức đề xuất laptop có thời lượng pin dài hơn.

→ **Thông kê:** Theo Forbes, các nền tảng áp dụng Sentiment Analysis vào recommendation tăng **15-25% doanh thu** từ gợi ý cá nhân hóa.

4. Kiến Trúc Microservice: Nền Tảng Linh Hoạt Cho AI Trong E-commerce

Để triển khai hiệu quả các mô hình AI, kiến trúc **microservice** đã chứng minh tính ưu việt:

- **AI Service độc lập:**

- Xử lý riêng các tác vụ như **phân tích cảm xúc, nhận diện hình ảnh** mà không ảnh hưởng đến hệ thống chính.

- **Khả năng mở rộng:**

- Dễ dàng thêm service mới (ví dụ: **dịch vụ phân tích xu hướng**) mà không cần refactor toàn bộ codebase.

- **Tích hợp đa nền tảng:**

- Kết nối với **dịch vụ Review** để lấy dữ liệu đánh giá, **dịch vụ User** để cá nhân hóa, và **dịch vụ Product** để cập nhật đề xuất.

→ **Ví dụ:** Tiki áp dụng microservice để triển khai **Real-time Sentiment Analysis**, xử lý 500.000 request/ngày với độ trễ dưới 200ms.

5. Thách Thức và Giải Pháp Khi Triển Khai AI

Dù mang lại nhiều lợi ích, việc ứng dụng AI trong e-commerce vẫn tồn tại một số thách thức:

- **Xử lý ngôn ngữ tự nhiên phức tạp:**
 - **Giải pháp:** Fine-tuning BERT trên dataset tiếng Việt, kết hợp **Word2Vec** để hiểu tiếng lóng, phương ngữ.
- **Chi phí tính toán cao:**
 - **Giải pháp:** Sử dụng **Cloud AI** (Google Vertex AI, AWS SageMaker) để giảm chi phí vận hành.
- **Bảo mật dữ liệu khách hàng:**
 - **Giải pháp:** Mã hóa dữ liệu bằng **GDPR-compliant encryption** và chỉ sử dụng dữ liệu ẩn danh.

6. Xu Hướng Tương Lai: AI Sẽ Định Hình Lại Thương Mại Điện Tử

- **Multimodal AI:** Kết hợp **text + hình ảnh + giọng nói** để phân tích toàn diện (ví dụ: phát hiện sản phẩm giả qua ảnh review).
- **AI tự động điều chỉnh giá:** Dựa trên cảm xúc khách hàng (ví dụ: giảm giá nếu nhiều người cho rằng sản phẩm "đắt").
- **Generative AI cho cá nhân hóa:** Tự động tạo mô tả sản phẩm hoặc email marketing phù hợp với từng phân khúc khách hàng.