HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN 1

*****

Assigment 2: Ecom Rest

Sinh viên: Nguyễn Mạnh Hùng

Mã SV: B21DCCN412

Lớp: D21CNPM2

Giảng viên: PGS.TS. Trần Đình Quế

Hà Nội, 2025

MỤC LỤC

Câu hỏi 1: REST

1.1. Giới thiệu về REST

Trong lĩnh vực phát triển phần mềm hiện đại, REST (Representational State Transfer) đóng vai trò quan trọng trong thiết kế và triển khai API (Application Programming Interface). Mục tiêu chính của phần này là tìm hiểu REST là gì, cách nó hoạt động cũng như vai trò của nó trong việc xây dựng các hệ thống phần mềm, đặc biệt là trong lĩnh vực thương mại điện tử. REST giúp các hệ thống giao tiếp hiệu quả thông qua giao thức HTTP, hỗ trợ khả năng mở rộng, tính linh hoạt và dễ dàng tích hợp giữa các dịch vụ khác nhau.

REST là một phong cách kiến trúc phần mềm dành cho việc thiết kế các API trên nền tảng web. REST không phải là một giao thức hay công nghệ cụ thể, mà là một tập hợp các nguyên tắc và ràng buộc nhằm đảm bảo rằng các hệ thống có thể giao tiếp với nhau theo cách đơn giản, thống nhất và có hiệu suất cao. Các API tuân theo kiến trúc REST được gọi là RESTful API.

Trong hệ thống thương mại điện tử, REST API đóng vai trò quan trọng trong việc kết nối giữa các thành phần như khách hàng, giỏ hàng, đơn hàng, thanh toán và vận chuyển. Chẳng hạn, một REST API có thể được sử dụng để truy xuất danh sách sản phẩm từ cơ sở dữ liệu, cho phép người dùng thêm sản phẩm vào giỏ hàng, thực hiện thanh toán và theo dõi tình trạng giao hàng. Nhờ vào sự đơn giản và khả năng mở rộng cao, REST trở thành một lựa chọn phổ biến trong việc phát triển các hệ thống thương mại điện tử hiện đại.

1.2. REST là gì?

REST (Representational State Transfer) là một phong cách kiến trúc được sử dụng để thiết kế API (Application Programming Interface) dựa trên giao thức HTTP. REST không phải là một tiêu chuẩn chính thức mà là một tập hợp các nguyên tắc giúp xây dựng hệ thống phân tán dễ dàng mở rộng, bảo trì và tối ưu hiệu suất. Các API được xây dựng theo phong cách REST được gọi là RESTful API, và chúng hoạt động thông qua các tài nguyên (resources) được định danh bằng URL.

Stateless (Không lưu trạng thái): Mỗi yêu cầu từ client đến server đều độc lập, tức là server không lưu trữ thông tin trạng thái của client giữa các lần gọi API. Điều này giúp hệ thống dễ mở rộng và ít tiêu tốn tài nguyên trên server.

Sử dụng các phương thức HTTP phổ biến: REST API sử dụng các phương thức HTTP tiêu chuẩn để thao tác với tài nguyên:

- `GET`: Lấy dữ liệu từ server (ví dụ: lấy danh sách sản phẩm).
- `POST`: Gửi dữ liệu mới lên server (ví dụ: tạo đơn hàng mới).
- `PUT`: Cập nhật tài nguyên hiện có (ví dụ: chỉnh sửa thông tin sản phẩm).

- `DELETE`: Xóa tài nguyên khỏi server (ví dụ: xóa đơn hàng).

Trả về dữ liệu dưới dạng JSON/XML: REST API thường sử dụng JSON (JavaScript Object Notation) hoặc XML (Extensible Markup Language) để trao đổi dữ liệu. JSON phổ biến hơn do cú pháp đơn giản, dễ đọc và nhẹ hơn XML.

Tiêu chí	REST	SOAP
Cấu trúc	Đơn giản, sử dụng HTTP	Phức tạp hơn, dựa trên XML
Giao thức	Hoạt động trên HTTP	Dựa trên nhiều giao thức (HTTP, SMTP, TCP,...)
Tốc độ	Nhanh, ít tốn tài nguyên	Chậm hơn do sử dụng XML và nhiều ràng buộc bảo mật
Dễ mở rộng	Dễ dàng mở rộng do tính stateless	Phức tạp hơn do có stateful
Hỗ trợ định dạng dữ liệu	JSON, XML	XML

REST API được ưa chuộng hơn SOAP trong các hệ thống web hiện đại do tính linh hoạt, đơn giản và hiệu suất cao. Trong khi SOAP vẫn được sử dụng trong các hệ thống yêu cầu bảo mật cao (như giao dịch tài chính), REST là lựa chọn hàng đầu cho các ứng dụng web, đặc biệt là trong thương mại điện tử và microservices.

1.3. Tại sao cần dùng REST?

Đơn giản, dễ triển khai

- REST sử dụng các phương thức HTTP chuẩn như `GET`, `POST`, `PUT`, `DELETE`, giúp các nhà phát triển dễ dàng triển khai mà không cần tạo thêm giao thức phức tạp.
- Cấu trúc API RESTful rõ ràng và dễ đọc, giúp việc tích hợp giữa các hệ thống trở nên đơn giản hơn.

Hỗ trợ mở rộng hệ thống (Scalability)

- Do tính chất **stateless** (không lưu trạng thái), REST giúp hệ thống có thể mở rộng dễ dàng bằng cách thêm nhiều máy chủ xử lý mà không bị phụ thuộc vào dữ liệu phiên (session data).
- Khi lượng truy cập tăng cao, hệ thống có thể phân tán tải (load balancing) một cách hiệu quả mà không làm giảm hiệu suất.

Tương thích đa nền tảng (Web, Mobile)

- REST API hoạt động trên giao thức HTTP, có thể được sử dụng bởi bất kỳ nền tảng nào có thể gửi yêu cầu HTTP, bao gồm ứng dụng web, mobile (Android, iOS), hoặc thậm chí các thiết bị IoT.
- Sử dụng định dạng dữ liệu JSON giúp API nhẹ và dễ xử lý trên nhiều loại thiết bị khác nhau.

Ví dụ thực tế: API lấy danh sách sản phẩm trong e-commerce: Giả sử chúng ta đang xây dựng một hệ thống thương mại điện tử. Một REST API có thể cung cấp danh sách sản phẩm để hiển thị trên trang web hoặc ứng dụng di động.

Yêu cầu API: Client gửi yêu cầu lấy danh sách sản phẩm:

```
GET /api/products
Host: example.com
Content-Type: application/json
```

Phản hồi từ server: Server trả về danh sách sản phẩm dưới dạng JSON

```
[
    {
        "id": 1,
        "name": "Laptop ASUS ROG",
        "price": 25000000,
        "stock": 10
    },
    {
        "id": 2,
        "name": "iPhone 14 Pro Max",
        "price": 32000000,
        "stock": 5
    }
]
```

Tại sao dùng REST trong trường hợp này?

- API dễ sử dụng: Client chỉ cần gửi yêu cầu `GET /api/products` để nhận dữ liệu.
- Hỗ trợ đa nền tảng: Dữ liệu JSON có thể được xử lý trên cả web và mobile app.
- Dễ mở rộng: Khi hệ thống có thêm nhiều sản phẩm, API vẫn hoạt động ổn định mà không cần thay đổi cấu trúc.

1.4. Các bước xây dựng REST trong Django

1.4.1. Bước 1: Cài đặt môi trường

- Yêu cầu: Cài đặt Python (>=3.7), Django, Django REST Framework (DRF).

```
pip install django djangorestframework
```

- Khởi tạo dự án:

```
django-admin startproject myproject
cd myproject
python manage.py startapp myapp
```

1.4.2. Bước 2: Tạo Model

- Ví dụ model Product:

```
from django.db import models

class Product(models.Model):
    name = models.CharField(max_length=255)
    price = models.DecimalField(max_digits=10, decimal_places=2)

    def __str__(self):
        return self.name
```

- Chạy migration để tạo bảng trong DB:

```
python manage.py makemigrations
python manage.py migrate
```

1.4.3. Bước 3: Tạo Serializer

- Dùng `serializers.ModelSerializer` để chuyển đổi Model → JSON:

```
from rest_framework import serializers
from .models import Product

class ProductSerializer(serializers.ModelSerializer):
    class Meta:
        model = Product
        fields = '__all__'
```

1.4.4. Bước 4: Tạo API View

```python
from rest_framework.views import APIView
from rest_framework.response import Response
from .models import Product
from .serializers import ProductSerializer

class ProductListView(APIView):
    def get(self, request):
        products = Product.objects.all()
        serializer = ProductSerializer(products, many=True)
        return Response(serializer.data)
```

1.4.5. Bước 5: Định tuyến URL

```python
from django.urls import path
from .views import ProductListView

urlpatterns = [
    path('products/', ProductListView.as_view(), name='product-list'),
]
```

1.5. Serialize là gì? Tại sao cần nó?

Serialization là quá trình chuyển đổi dữ liệu từ đối tượng Python (như Model trong Django) sang định dạng JSON hoặc XML để có thể truyền qua API. Ngược lại, **Deserialization** là quá trình chuyển đổi dữ liệu từ JSON/XML về đối tượng Python để xử lý hoặc lưu vào cơ sở dữ liệu.
Trong một hệ thống REST API, Serialization đóng vai trò quan trọng trong việc trao đổi dữ liệu giữa server và client. Khi server gửi dữ liệu, nó cần chuyển đổi từ đối tượng Model sang JSON để client có thể đọc được. Ngược lại, khi client gửi dữ liệu đến server thông qua các phương thức như POST hoặc PUT, server cần chuyển JSON thành đối tượng Python để thực hiện lưu trữ hoặc xử lý.

Ví dụ, trong một ứng dụng thương mại điện tử, nếu hệ thống có một danh sách sản phẩm, mỗi sản phẩm có các thuộc tính như **id, tên, giá**, thì API cần chuyển đổi danh sách này thành JSON để client có thể hiển thị thông tin trên website hoặc ứng dụng di động. Chẳng hạn, một sản phẩm có thể được serialize thành dữ liệu JSON như:

{"id": 1, "name": "Book", "price": 10}

Quá trình này giúp API có thể hoạt động hiệu quả trên nhiều nền tảng khác nhau, từ trình duyệt web đến ứng dụng di động, mà không phụ thuộc vào công nghệ backend.

1.6. Sử dụng Visual Paradigm để biểu diễn REST

Visual Paradigm (VP) là một công cụ mạnh mẽ hỗ trợ thiết kế mô hình UML, giúp lập trình viên và kiến trúc sư hệ thống dễ dàng biểu diễn và hiểu các thành phần trong một ứng dụng REST. Công cụ này cung cấp nhiều loại sơ đồ khác nhau, giúp minh họa cách REST API hoạt động trong hệ thống.

Cách sử dụng Visual Paradigm để mô tả REST API

1. **Sơ đồ Use Case**

    - Sơ đồ này giúp xác định các tác nhân (actors) và chức năng chính mà hệ thống cung cấp.
    - Ví dụ: Đối với API lấy danh sách sản phẩm, tác nhân chính là **Người dùng (User)** hoặc **Ứng dụng khách (Client App)**, còn hệ thống cung cấp một **Use Case** là **Lấy danh sách sản phẩm** qua API.
2. **Sơ đồ Sequence**

    - Sơ đồ này thể hiện luồng dữ liệu và cách các thành phần tương tác với nhau theo trình tự thời gian.
    - Ví dụ: Khi client gửi yêu cầu **GET /products/**, luồng dữ liệu có thể được mô tả như sau:
        - **Client** gửi yêu cầu HTTP **GET /products/** đến server.
        - **Server** kiểm tra quyền truy cập và truy vấn dữ liệu từ cơ sở dữ liệu.
        - **Server** trả về danh sách sản phẩm dưới dạng JSON.
        - **Client** hiển thị dữ liệu lên giao diện người dùng.

Bằng cách sử dụng Visual Paradigm để mô hình hóa REST API, nhóm phát triển có thể dễ dàng trao đổi và hiểu được cách các thành phần tương tác với nhau, giúp việc thiết kế và triển khai hệ thống trở nên rõ ràng hơn.

1.7. Ví dụ thực tế (3 ví dụ)

REST API được sử dụng rộng rãi trong các hệ thống web để cung cấp dữ liệu và xử lý tương tác giữa client và server. Dưới đây là ba ví dụ thực tế về việc triển khai REST API trong một ứng dụng thương mại điện tử sử dụng Django REST Framework (DRF).

Ví dụ 1: API lấy danh sách sản phẩm

API này cho phép client lấy danh sách sản phẩm từ hệ thống. Khi gửi yêu cầu **GET /products/**, server sẽ truy vấn cơ sở dữ liệu và trả về danh sách sản phẩm dưới dạng JSON. Điều này giúp các ứng dụng web hoặc mobile có thể hiển thị danh sách sản phẩm một cách linh hoạt.

Ví dụ 2: API thêm sản phẩm vào giỏ hàng

Trong một hệ thống thương mại điện tử, người dùng có thể thêm sản phẩm vào giỏ hàng. Khi client gửi yêu cầu **POST /cart/add/** cùng với dữ liệu sản phẩm dưới dạng JSON, server sẽ xử lý yêu cầu, kiểm tra sản phẩm hợp lệ và lưu vào giỏ hàng của người dùng. Điều này giúp đảm bảo giỏ hàng có thể được cập nhật và đồng bộ giữa các thiết bị.

Ví dụ 3: API đăng ký người dùng

Việc đăng ký tài khoản là một chức năng quan trọng trong các hệ thống web. Khi người dùng gửi yêu cầu **POST /register/** với thông tin đăng ký như tên, email, và mật khẩu, server sẽ sử dụng Serializer để kiểm tra dữ liệu, lưu vào cơ sở dữ liệu và trả về phản hồi xác nhận. API này giúp người dùng tạo tài khoản một cách nhanh chóng và bảo mật.

Ba ví dụ trên minh họa cách REST API hoạt động trong các hệ thống thực tế, giúp ứng dụng web hoạt động hiệu quả và hỗ trợ tốt cho nhiều loại client khác nhau như trình duyệt web, ứng dụng di động hoặc hệ thống bên thứ ba.

1.8. Kết luận

REST là một nền tảng quan trọng trong thiết kế API, đặc biệt là trong các hệ thống microservice hiện đại. Với khả năng đơn giản, dễ triển khai và mở rộng, REST đã trở thành tiêu chuẩn phổ biến để xây dựng các hệ thống web và dịch vụ trực tuyến.
Trong một hệ thống thương mại điện tử, REST đóng vai trò kết nối giữa các module như quản lý sản phẩm, giỏ hàng, đơn hàng và người dùng. Nhờ REST API, các ứng dụng web, mobile và hệ thống bên thứ ba có thể tương tác dễ dàng với backend, đảm bảo dữ liệu được đồng bộ và hoạt động trơn tru.

Với những lợi ích vượt trội, REST vẫn là một lựa chọn tối ưu khi thiết kế API cho các ứng dụng hiện đại, giúp nâng cao trải nghiệm người dùng và tối ưu hiệu suất hệ thống.

Câu hỏi 2: Thiết kế hệ thống thương mại điện tử trong Django (17 trang)

2.1. Giới thiệu hệ thống

Hệ thống thương mại điện tử được thiết kế theo mô hình microservice nhằm đảm bảo tính linh hoạt, dễ mở rộng và khả năng bảo trì cao. Việc sử dụng Django và REST API giúp hệ thống có thể cung cấp các dịch vụ nhanh chóng, đồng thời hỗ trợ tích hợp với nhiều nền tảng khác nhau như web, mobile.

Tổng quan về hệ thống

Hệ thống bao gồm nhiều module chính, mỗi module đảm nhận một nhiệm vụ riêng biệt:

- **Customer**: Quản lý thông tin khách hàng, đăng ký, đăng nhập và quyền truy cập.
- **Cart**: Xử lý giỏ hàng, thêm/xóa sản phẩm, cập nhật số lượng.
- **Order**: Quản lý đơn hàng, xử lý trạng thái đơn hàng từ khi đặt hàng đến khi hoàn tất.
- **Shipping**: Quản lý vận chuyển, theo dõi trạng thái giao hàng.
- **Paying**: Xử lý thanh toán trực tuyến qua các cổng thanh toán.
- **Items**: Quản lý sản phẩm, danh mục, giá cả và thông tin liên quan.

Công nghệ sử dụng

- **Django**: Framework chính để xây dựng backend của hệ thống.
- **REST API**: Giao tiếp giữa các dịch vụ microservice.
- **Visual Paradigm**: Công cụ hỗ trợ thiết kế UML, giúp trực quan hóa mô hình hệ thống.

Cơ sở dữ liệu

Hệ thống có thể kết hợp nhiều loại cơ sở dữ liệu để tối ưu hiệu suất:

- **MySQL**: Dùng cho dữ liệu quan hệ, chẳng hạn như thông tin đơn hàng, khách hàng.
- **MongoDB**: Lưu trữ dữ liệu phi cấu trúc như lịch sử duyệt sản phẩm.
- **PostgreSQL**: Một lựa chọn linh hoạt để quản lý dữ liệu có quan hệ và yêu cầu xử lý truy vấn mạnh mẽ.

Hệ thống e-commerce này được thiết kế để hỗ trợ nhiều người dùng, đảm bảo tốc độ xử lý nhanh và khả năng mở rộng cao khi số lượng giao dịch tăng lên.

2.2. Kiến trúc tổng thể

Hệ thống thương mại điện tử được thiết kế theo kiến trúc **microservice**, trong đó mỗi module hoạt động như một dịch vụ độc lập, có cơ sở dữ liệu riêng và giao tiếp với các module khác

thông qua **REST API**. Kiến trúc này giúp hệ thống dễ dàng mở rộng, bảo trì và triển khai từng phần mà không ảnh hưởng đến toàn bộ hệ thống.
Các dịch vụ chính bao gồm:

- **Customer Service**: Xử lý đăng ký, đăng nhập, quản lý tài khoản khách hàng.
- **Cart Service**: Quản lý giỏ hàng, thêm/xóa sản phẩm.
- **Order Service**: Xử lý đơn hàng, quản lý trạng thái đơn hàng.
- **Shipping Service**: Quản lý vận chuyển, theo dõi giao hàng.
- **Payment Service**: Xử lý thanh toán qua nhiều phương thức khác nhau.
- **Product Service**: Quản lý sản phẩm, danh mục, thông tin giá cả.

Sơ đồ kiến trúc

Trong **Visual Paradigm (VP)**, sơ đồ hệ thống có thể biểu diễn như sau:

- **Client Layer**: Gồm ứng dụng web, mobile, hoặc các nền tảng bên thứ ba sử dụng API.
- **API Gateway**: Là điểm truy cập duy nhất cho client, định tuyến yêu cầu đến các dịch vụ phù hợp.
- **Service Layer**: Bao gồm các microservice độc lập, xử lý nghiệp vụ chính của hệ thống.
- **Database Layer**: Mỗi microservice có thể sử dụng một loại cơ sở dữ liệu phù hợp, chẳng hạn như MySQL cho dữ liệu quan hệ, MongoDB cho dữ liệu phi cấu trúc.

Kết nối giữa các module

Các module trong hệ thống giao tiếp với nhau thông qua **REST API**, giúp hệ thống linh hoạt trong việc phát triển và mở rộng. Mỗi service chỉ cần đảm nhiệm một nhiệm vụ duy nhất, giúp giảm tải cho các thành phần khác và đảm bảo tính ổn định.

Với mô hình này, hệ thống có thể dễ dàng mở rộng khi cần thiết, chẳng hạn như tăng thêm server cho một microservice cụ thể khi lưu lượng truy cập tăng cao.

2.3. Thiết kế module Customer

2.3.1. Chức năng

Module **Customer** là thành phần quan trọng trong hệ thống thương mại điện tử, chịu trách nhiệm quản lý thông tin người dùng và cung cấp các chức năng liên quan đến tài khoản khách hàng. Hệ thống phân loại khách hàng thành 4 nhóm chính:

- **Khách hàng thường**: Người dùng phổ thông, không có quyền lợi đặc biệt.
- **Khách hàng VIP**: Được hưởng ưu đãi và chính sách giảm giá đặc biệt.
- **Khách hàng doanh nghiệp**: Các công ty, tổ chức có nhu cầu mua hàng số lượng lớn.
- **Khách hàng quốc tế**: Người dùng đến từ các quốc gia khác, có thể yêu cầu vận chuyển xuyên biên giới.

Các chức năng chính của module Customer bao gồm:

1. **Đăng ký tài khoản**: Người dùng mới có thể tạo tài khoản bằng email và mật khẩu.
2. **Đăng nhập**: Xác thực người dùng bằng thông tin đăng nhập.
3. **Cập nhật thông tin cá nhân**: Cho phép khách hàng chỉnh sửa thông tin như tên, số điện thoại, địa chỉ.
4. **Xem lịch sử đơn hàng**: Khách hàng có thể tra cứu các đơn hàng đã đặt trước đó.
5. **Xóa tài khoản**: Người dùng có thể yêu cầu xóa tài khoản khỏi hệ thống.
6. **Đổi mật khẩu**: Cung cấp cơ chế thay đổi hoặc khôi phục mật khẩu khi cần thiết.
7. **Nhận thông báo**: Gửi thông báo về đơn hàng, khuyến mãi, hoặc cập nhật từ hệ thống.

2.3.2. Cơ sở dữ liệu

Module Customer sử dụng **MySQL** để lưu trữ dữ liệu khách hàng. Dữ liệu được tổ chức thành bảng **Customer** với các trường:

- **id** (Primary Key) – Định danh duy nhất cho mỗi khách hàng.
- **name** – Tên đầy đủ của khách hàng.
- **email** – Địa chỉ email, dùng để đăng nhập.
- **type** – Phân loại khách hàng (Thường, VIP, Doanh nghiệp, Quốc tế).
- **password** – Mật khẩu được mã hóa để bảo mật thông tin.

2.3.3. Sơ đồ Use Case

Trong **Visual Paradigm**, sơ đồ **Use Case** giúp mô tả các hành vi chính của khách hàng trong hệ thống. Một số Use Case quan trọng:

- **Đăng ký tài khoản**: Người dùng nhập thông tin cá nhân và hệ thống tạo tài khoản mới.
- **Xem lịch sử đơn hàng**: Người dùng truy cập danh sách các đơn hàng đã mua trước đó.

2.3.4. Sơ đồ Class

Sơ đồ **Class Diagram** thể hiện cấu trúc dữ liệu của module Customer.

- **Class Customer** có các thuộc tính:
    - `id` (int)
    - `name` (string)
    - `email` (string)
    - `type` (enum)

    - `password` (hashed string)

- Các phương thức chính của **Customer**:
    - `register()` – Tạo tài khoản mới.
    - `updateInfo()` – Cập nhật thông tin cá nhân.

Mối quan hệ giữa các class:

- **Một Customer có thể có nhiều Order**.
- **Customer liên kết với các module khác như Cart và Payment** để xử lý đơn hàng và thanh toán.

2.4. Thiết kế module Items

2.4.1. Chức năng

Module **Items** chịu trách nhiệm quản lý danh mục sản phẩm trong hệ thống thương mại điện tử. Người dùng có thể thao tác với sản phẩm thông qua các chức năng sau:

1. **Thêm sản phẩm**: Quản trị viên hoặc người bán có thể thêm sản phẩm mới vào hệ thống.
2. **Xóa sản phẩm**: Khi một sản phẩm không còn kinh doanh, hệ thống có thể xóa nó khỏi danh mục.
3. **Cập nhật giá**: Giá sản phẩm có thể thay đổi tùy theo thời điểm, chương trình khuyến mãi hoặc điều chỉnh từ nhà cung cấp.
4. **Tìm kiếm sản phẩm**: Cho phép khách hàng nhập từ khóa để tìm sản phẩm mong muốn.
5. **Lọc theo loại sản phẩm**: Người dùng có thể lọc sản phẩm theo danh mục như sách, điện thoại, quần áo, giày dép.
6. **Xem chi tiết sản phẩm**: Hiển thị đầy đủ thông tin về sản phẩm như mô tả, giá cả, đánh giá, hình ảnh.
7. **Thêm vào danh sách yêu thích**: Khách hàng có thể lưu sản phẩm vào danh sách yêu thích để theo dõi sau này.

2.4.2. Cơ sở dữ liệu

Module **Items** sử dụng **MongoDB** để lưu trữ dữ liệu sản phẩm, phù hợp với hệ thống thương mại điện tử có danh mục sản phẩm phong phú và linh hoạt.

Dữ liệu được lưu dưới dạng **Collection Items** với các trường:

- **id** (ObjectID) – Định danh duy nhất cho mỗi sản phẩm.
- **name** (string) – Tên sản phẩm.
- **type** (string) – Loại sản phẩm (book, mobile, clothes, shoes).

- **price** (float) – Giá của sản phẩm.

MongoDB cho phép linh hoạt trong việc lưu trữ các loại sản phẩm khác nhau mà không cần thay đổi cấu trúc dữ liệu.

2.4.3. Sơ đồ Use Case

Sơ đồ **Use Case** trong **Visual Paradigm** mô tả cách người dùng tương tác với module Items. Một trong những Use Case quan trọng là **"Tìm kiếm sản phẩm"**, trong đó:

- Người dùng nhập từ khóa tìm kiếm.
- Hệ thống trả về danh sách sản phẩm phù hợp.
- Người dùng có thể chọn sản phẩm để xem chi tiết.

2.4.4. Sơ đồ Class

Sơ đồ **Class Diagram** thể hiện cấu trúc dữ liệu của module Items:

- **Class Item** chứa các thuộc tính:
  - `id` (ObjectID)
  - `name` (string)
  - `type` (enum: book, mobile, clothes, shoes)
  - `price` (float)

Các phương thức của **Item**:

- `addItem()` – Thêm sản phẩm mới vào hệ thống.
- `updatePrice()` – Cập nhật giá sản phẩm.

Kế thừa từ class **Item**, hệ thống có thể mở rộng thành các class con:

- **Book** – Sản phẩm là sách, có thêm thuộc tính `author`, `publisher`.
- **Mobile** – Điện thoại, có thêm `brand`, `ram`, `storage`.
- **Clothes** – Quần áo, có thêm `size`, `color`, `material`.
- **Shoes** – Giày dép, có thêm `size`, `brand`.

2.5. Thiết kế module Cart

2.5.1. Chức năng

Module **Cart (Giỏ hàng)** chịu trách nhiệm lưu trữ danh sách sản phẩm mà khách hàng chọn mua trước khi tiến hành thanh toán. Các chức năng chính bao gồm:

1. **Thêm sản phẩm vào giỏ hàng**: Khi người dùng chọn một sản phẩm, hệ thống sẽ thêm nó vào giỏ hàng của họ.
2. **Xóa sản phẩm khỏi giỏ hàng**: Người dùng có thể loại bỏ sản phẩm không còn nhu cầu mua.
3. **Cập nhật số lượng sản phẩm**: Nếu người dùng muốn thay đổi số lượng sản phẩm trong giỏ, họ có thể điều chỉnh.
4. **Xem giỏ hàng**: Hiển thị danh sách các sản phẩm hiện có trong giỏ hàng, bao gồm tên, giá, số lượng.
5. **Tính tổng tiền**: Hệ thống tự động tính toán tổng chi phí của các sản phẩm trong giỏ hàng.
6. **Áp dụng mã giảm giá**: Người dùng có thể nhập mã giảm giá để được hưởng ưu đãi trên tổng hóa đơn.
7. **Lưu giỏ hàng**: Giúp khách hàng có thể quay lại tiếp tục mua sắm mà không mất dữ liệu giỏ hàng.

2.5.2. Cơ sở dữ liệu

Module **Cart** sử dụng **PostgreSQL** để lưu trữ thông tin giỏ hàng do tính ổn định và hỗ trợ tốt cho dữ liệu có cấu trúc.

Cấu trúc bảng **Cart** gồm các trường sau:

- **id** (UUID) – Định danh duy nhất của giỏ hàng.
- **customer_id** (UUID) – ID khách hàng sở hữu giỏ hàng.
- **items** (JSON/List) – Danh sách sản phẩm trong giỏ, bao gồm:
    - `product_id`: ID sản phẩm.
    - `quantity`: Số lượng sản phẩm.
    - `price`: Giá mỗi sản phẩm tại thời điểm thêm vào giỏ.

Lưu danh sách sản phẩm dưới dạng **JSON** giúp linh hoạt trong việc lưu trữ và xử lý dữ liệu.

2.5.3. Sơ đồ Use Case

Sơ đồ **Use Case** trong **Visual Paradigm** mô tả cách khách hàng tương tác với giỏ hàng. Một trong những Use Case quan trọng là **"Thêm sản phẩm vào giỏ"**, gồm các bước sau:

- Người dùng chọn sản phẩm và nhấn **"Thêm vào giỏ"**.
- Hệ thống kiểm tra xem sản phẩm đã có trong giỏ chưa:
    - Nếu có, tăng số lượng.
    - Nếu chưa, thêm sản phẩm vào giỏ với số lượng ban đầu là 1.
- Hệ thống cập nhật giỏ hàng và hiển thị thông báo thành công.

2.5.4. Sơ đồ Class

Sơ đồ **Class Diagram** thể hiện cấu trúc của module **Cart** như sau:

- **Class Cart**:
    - **Thuộc tính**:
        - `id` (UUID)
        - `customer_id` (UUID)
        - `items` (List of JSON objects)
    - **Phương thức**:
        - `addItem(product_id, quantity)`: Thêm sản phẩm vào giỏ hoặc cập nhật số lượng.
        - `removeItem(product_id)`: Xóa sản phẩm khỏi giỏ hàng.
        - `calculateTotal()`: Tính tổng tiền của giỏ hàng.
        - `applyDiscount(code)`: Áp dụng mã giảm giá.
        - `saveCart()`: Lưu giỏ hàng vào cơ sở dữ liệu.

2.6. Thiết kế module Order

2.6.1. Chức năng

Module **Order (Đơn hàng)** đóng vai trò quan trọng trong hệ thống thương mại điện tử, giúp xử lý các giao dịch mua bán. Các chức năng chính bao gồm:

1. **Tạo đơn hàng**: Khi người dùng xác nhận mua hàng, hệ thống tạo đơn hàng từ giỏ hàng.
2. **Hủy đơn hàng**: Nếu khách hàng thay đổi ý định hoặc có vấn đề phát sinh, họ có thể hủy đơn hàng trước khi được xử lý.
3. **Xem trạng thái đơn hàng**: Khách hàng có thể kiểm tra đơn hàng đang ở trạng thái nào (Đang xử lý, Đang giao, Hoàn thành, Đã hủy).

4. **Cập nhật trạng thái đơn hàng**: Hệ thống hoặc nhân viên quản trị có thể thay đổi trạng thái đơn hàng theo tiến trình xử lý.
5. **Tính tổng tiền**: Hệ thống tự động tính tổng giá trị đơn hàng dựa trên danh sách sản phẩm, số lượng, chi phí vận chuyển và giảm giá (nếu có).
6. **Gửi thông báo**: Khi đơn hàng thay đổi trạng thái, hệ thống sẽ gửi thông báo qua email hoặc ứng dụng.
7. **Lưu lịch sử đơn hàng**: Mọi giao dịch được lưu trữ để hỗ trợ khách hàng theo dõi đơn hàng và khiếu nại nếu cần.

2.6.2. Cơ sở dữ liệu

Module **Order** sử dụng **PostgreSQL** do yêu cầu về dữ liệu có cấu trúc rõ ràng và khả năng mở rộng.

Cấu trúc bảng **Order** gồm các trường sau:

- **id** (UUID) – Định danh duy nhất của đơn hàng.
- **customer_id** (UUID) – ID khách hàng đặt đơn hàng.
- **total** (Decimal) – Tổng tiền đơn hàng.
- **status** (Enum) – Trạng thái đơn hàng (`Pending`, `Processing`, `Shipped`, `Completed`, `Cancelled`).

2.6.3. Sơ đồ Use Case

Sơ đồ **Use Case** trong **Visual Paradigm** minh họa cách khách hàng và hệ thống tương tác với đơn hàng.
Ví dụ: **"Tạo đơn hàng"**

- Người dùng bấm **"Thanh toán"**.
- Hệ thống kiểm tra giỏ hàng:
  - Nếu giỏ hàng trống, thông báo lỗi.
  - Nếu hợp lệ, hệ thống tạo đơn hàng với trạng thái `Pending`.
- Hệ thống lưu đơn hàng vào cơ sở dữ liệu và hiển thị thông báo đặt hàng thành công.

2.6.4. Sơ đồ Class

Sơ đồ **Class Diagram** mô tả các đối tượng chính trong module **Order**:

- **Class Order**:
  - **Thuộc tính**:
    - `id` (UUID)

    - `customer_id` (UUID)
    - `total` (Decimal)
    - `status` (Enum)
- **Phương thức**:
    - `createOrder(customer_id, cart_items)`: Tạo đơn hàng từ giỏ hàng.
    - `updateStatus(order_id, new_status)`: Cập nhật trạng thái đơn hàng.
    - `calculateTotal(order_id)`: Tính tổng tiền đơn hàng.
    - `sendNotification(order_id, message)`: Gửi thông báo đến khách hàng.
    - `saveOrderHistory(order_id)`: Lưu lịch sử đơn hàng.

2.7. Thiết kế module Shipping

2.7.1. Chức năng

Module **Shipping (Vận chuyển)** chịu trách nhiệm quản lý quá trình giao hàng, từ khi đơn hàng được xác nhận đến khi giao cho khách. Các chức năng chính bao gồm:

1. **Chọn phương thức vận chuyển**: Người dùng có thể chọn giữa các hình thức như giao hàng tiêu chuẩn, giao hàng nhanh hoặc nhận tại cửa hàng.
2. **Tính phí vận chuyển**: Hệ thống tính toán phí dựa trên khoảng cách, trọng lượng, loại hàng và phương thức vận chuyển.
3. **Cập nhật trạng thái giao hàng**: Hệ thống hoặc nhân viên giao hàng có thể thay đổi trạng thái (`Processing`, `Shipped`, `Delivered`, `Cancelled`).
4. **Theo dõi đơn hàng**: Khách hàng có thể kiểm tra vị trí và trạng thái của đơn hàng theo thời gian thực.
5. **Hủy giao hàng**: Nếu đơn hàng bị hủy hoặc khách hàng yêu cầu, hệ thống sẽ xử lý việc hủy vận chuyển.
6. **Gửi thông báo**: Khi trạng thái giao hàng thay đổi, hệ thống gửi thông báo qua email hoặc tin nhắn.
7. **Lưu lịch sử giao hàng**: Mọi thông tin vận chuyển được ghi nhận để hỗ trợ truy vết khi cần.

2.7.2. Cơ sở dữ liệu

Module **Shipping** sử dụng **PostgreSQL** để lưu trữ thông tin vận chuyển.

Cấu trúc bảng **Shipping** gồm:

- **id** (UUID) – Định danh duy nhất của vận chuyển.
- **order_id** (UUID) – ID đơn hàng tương ứng.
- **status** (Enum) – Trạng thái vận chuyển (`Processing`, `Shipped`, `Delivered`, `Cancelled`).
- **cost** (Decimal) – Chi phí vận chuyển.

2.7.3. Sơ đồ Use Case

Sơ đồ **Use Case** trong **Visual Paradigm** minh họa quá trình **"Theo dõi đơn hàng"**:

- Khách hàng gửi yêu cầu kiểm tra trạng thái đơn hàng.
- Hệ thống truy vấn thông tin từ cơ sở dữ liệu và trả về trạng thái hiện tại (`Shipped`, `Delivered`, v.v.).
- Khách hàng nhận thông tin và có thể xem lộ trình giao hàng nếu có tích hợp dịch vụ theo dõi.

2.7.4. Sơ đồ Class

Sơ đồ **Class Diagram** mô tả các đối tượng chính trong module **Shipping**:

- **Class Shipping**:
  - **Thuộc tính**:
    - `id` (UUID)
    - `order_id` (UUID)
    - `status` (Enum)
    - `cost` (Decimal)
  - **Phương thức**:

- `updateStatus(shipping_id, new_status)`: Cập nhật trạng thái vận chuyển.
- `calculateCost(order_id, shipping_method)`: Tính phí vận chuyển.
- `trackOrder(order_id)`: Lấy thông tin theo dõi đơn hàng.
- `sendNotification(order_id, message)`: Gửi thông báo về tình trạng giao hàng.
- `saveShippingHistory(order_id)`: Lưu lịch sử giao hàng.

2.8. Thiết kế module Paying

2.8.1. Chức năng

Module **Paying (Thanh toán)** đảm nhận việc xử lý các giao dịch tài chính của hệ thống thương mại điện tử. Các chức năng chính bao gồm:

1. **Chọn phương thức thanh toán**: Hỗ trợ nhiều hình thức thanh toán như thẻ tín dụng, ví điện tử (Momo, PayPal), chuyển khoản ngân hàng hoặc thanh toán khi nhận hàng (COD).
2. **Thực hiện thanh toán**: Xác thực thông tin và xử lý giao dịch với ngân hàng hoặc cổng thanh toán.
3. **Hoàn tiền**: Cho phép hoàn tiền trong trường hợp đơn hàng bị hủy hoặc yêu cầu trả hàng.
4. **Kiểm tra trạng thái thanh toán**: Cung cấp API để kiểm tra tình trạng giao dịch (`Pending`, `Completed`, `Failed`).
5. **Lưu lịch sử thanh toán**: Ghi nhận mọi giao dịch để phục vụ truy vết và báo cáo tài chính.
6. **Gửi hóa đơn**: Tự động tạo và gửi hóa đơn điện tử cho khách hàng qua email.
7. **Thông báo thanh toán**: Hệ thống gửi thông báo khi thanh toán thành công hoặc thất bại.

2.8.2. Cơ sở dữ liệu

Module **Paying** sử dụng **PostgreSQL** để lưu trữ thông tin thanh toán.

Cấu trúc bảng **Payment** gồm:

- **id** (UUID) – Định danh duy nhất của giao dịch thanh toán.
- **order_id** (UUID) – ID của đơn hàng liên quan.
- **amount** (Decimal) – Số tiền thanh toán.

- **status** (Enum) – Trạng thái thanh toán (`Pending`, `Completed`, `Failed`, `Refunded`).

2.8.3. Sơ đồ Use Case

Sơ đồ **Use Case** trong **Visual Paradigm** minh họa quá trình **"Thực hiện thanh toán"**:

- Khách hàng chọn phương thức thanh toán và gửi yêu cầu.
- Hệ thống kiểm tra thông tin đơn hàng và thực hiện giao dịch.
- Giao dịch được xác nhận thành công hoặc thất bại, và hệ thống cập nhật trạng thái tương ứng.
- Khách hàng nhận được thông báo và hóa đơn sau khi thanh toán hoàn tất.

2.8.4. Sơ đồ Class

Sơ đồ **Class Diagram** mô tả các đối tượng chính trong module **Paying**:

- **Class Payment**:
  - **Thuộc tính**:
    - `id` (UUID)
    - `order_id` (UUID)
    - `amount` (Decimal)
    - `status` (Enum)
  - **Phương thức**:
    - `processPayment(order_id, payment_method)`: Xử lý thanh toán cho đơn hàng.
    - `refund(payment_id)`: Hoàn tiền cho giao dịch bị hủy.
    - `checkStatus(payment_id)`: Kiểm tra trạng thái thanh toán.
    - `generateInvoice(order_id)`: Tạo và gửi hóa đơn cho khách hàng.
    - `sendPaymentNotification(order_id, message)`: Gửi thông báo thanh toán.

2.9. Kết nối REST giữa các module

Hệ thống thương mại điện tử được thiết kế theo kiến trúc **microservice**, trong đó các module hoạt động độc lập và giao tiếp với nhau thông qua **REST API**. Mỗi module cung cấp các API cụ thể để trao đổi dữ liệu với module khác và với client.

Ví dụ API

Dưới đây là một số API tiêu biểu thể hiện cách các module kết nối với nhau thông qua REST:

1. **Lấy danh sách sản phẩm**
    - **API:** `GET /items/`
    - **Luồng dữ liệu:**
        - **Client** gửi yêu cầu lấy danh sách sản phẩm.
        - **Module Items** xử lý và trả về danh sách sản phẩm dưới dạng JSON.
2. **Thêm sản phẩm vào giỏ hàng**
    - **API:** `POST /cart/add/`
    - **Luồng dữ liệu:**
        - **Client** gửi yêu cầu thêm sản phẩm vào giỏ.
        - **Module Cart** kiểm tra sản phẩm trong **Module Items**.
        - Nếu hợp lệ, sản phẩm được thêm vào giỏ hàng và lưu vào database.
3. **Tạo đơn hàng và xử lý thanh toán**
    - **API:** `POST /order/create/`
    - **Luồng dữ liệu:**
        - **Client** gửi yêu cầu tạo đơn hàng.
        - **Module Cart** xác nhận các sản phẩm trong giỏ và gửi yêu cầu đến **Module Order**.
        - **Module Order** tạo đơn hàng và gửi yêu cầu thanh toán đến **Module Paying**.
        - Nếu thanh toán thành công, đơn hàng được gửi đến **Module Shipping** để xử lý vận chuyển.
        - Hệ thống cập nhật trạng thái đơn hàng và thông báo cho khách hàng.

Sơ đồ Sequence trong Visual Paradigm

Trong **Visual Paradigm**, sơ đồ **Sequence Diagram** thể hiện luồng dữ liệu giữa các module. Ví dụ, sơ đồ cho quá trình đặt hàng có các bước như sau:

1. **Customer** gửi yêu cầu đặt hàng.
2. **Module Cart** xác nhận giỏ hàng và gửi yêu cầu đến **Module Order**.
3. **Module Order** tạo đơn hàng và gọi API từ **Module Paying** để xử lý thanh toán.
4. Nếu thanh toán thành công, **Module Shipping** nhận đơn hàng và lên kế hoạch giao hàng.
5. Hệ thống cập nhật trạng thái đơn hàng và thông báo đến khách hàng.

2.10. Triển khai code

Công cụ sử dụng

Hệ thống thương mại điện tử được triển khai bằng **Django** kết hợp với **Django REST Framework (DRF)** để xây dựng API. Mỗi module sử dụng hệ quản trị cơ sở dữ liệu phù hợp:

- **MySQL**: Lưu trữ thông tin khách hàng (Customer).
- **MongoDB**: Quản lý danh sách sản phẩm (Items).
- **PostgreSQL**: Lưu trữ giỏ hàng, đơn hàng, thanh toán và vận chuyển.

Các thư viện quan trọng:

- `djangorestframework` – Hỗ trợ xây dựng API RESTful.
- `mysql-connector-python` – Kết nối với MySQL.
- `pymongo` – Tương tác với MongoDB.
- `psycopg2` – Kết nối với PostgreSQL.

Ví dụ triển khai code

1. **Model Customer với MySQL**
    - Thông tin khách hàng được lưu trữ trong MySQL.
    - Model định nghĩa các trường như `id`, `name`, `email`, `password`, `type`

      (Thường, VIP, Doanh nghiệp, Quốc tế).
2. **API GET /items/ với MongoDB**
    - API này truy vấn danh sách sản phẩm từ MongoDB.
    - Kết quả trả về dưới dạng JSON chứa danh sách sản phẩm.
3. **API POST /cart/add/ với PostgreSQL**
    - Khi người dùng thêm sản phẩm vào giỏ hàng, API sẽ lưu dữ liệu vào PostgreSQL.

    - API kiểm tra sản phẩm hợp lệ trước khi lưu vào giỏ hàng.

Hệ thống kết hợp nhiều công nghệ để đảm bảo tính mở rộng và hiệu suất cao, giúp hệ thống thương mại điện tử hoạt động hiệu quả và ổn định.

2.11. Kết luận

Hệ thống thương mại điện tử được thiết kế theo kiến trúc microservice kết hợp với REST API, giúp chia nhỏ hệ thống thành các module độc lập như Customer, Items, Cart, Order, Shipping, Paying. Mỗi module có cơ sở dữ liệu riêng (MySQL, MongoDB, PostgreSQL) và giao tiếp với nhau thông qua API, đảm bảo tính linh hoạt và khả năng mở rộng của hệ thống.

Đánh giá

- **Ưu điểm:**
    - **Dễ mở rộng**: Mỗi module hoạt động độc lập, có thể nâng cấp hoặc thay thế mà không ảnh hưởng đến toàn bộ hệ thống.
    - **Tính linh hoạt cao**: Sử dụng nhiều công nghệ phù hợp với từng module, tối ưu hóa hiệu suất.
    - **Bảo mật tốt**: Phân tách dữ liệu theo từng module giúp hạn chế rủi ro bảo mật.
- **Hạn chế:**
    - **Phức tạp trong tích hợp**: Việc kết nối giữa các module qua REST API và quản lý nhiều hệ quản trị cơ sở dữ liệu khác nhau đòi hỏi sự cẩn thận trong thiết kế.
    - **Hiệu suất có thể bị ảnh hưởng**: Nếu không tối ưu API và cơ sở dữ liệu, hệ thống có thể gặp tình trạng quá tải khi lượng người dùng tăng cao.

Hệ thống e-commerce này là một giải pháp mạnh mẽ, có khả năng mở rộng và phù hợp cho các doanh nghiệp lớn. Tuy nhiên, để triển khai thực tế, cần có chiến lược quản lý API, tối ưu hóa cơ sở dữ liệu và đảm bảo hiệu suất hệ thống.