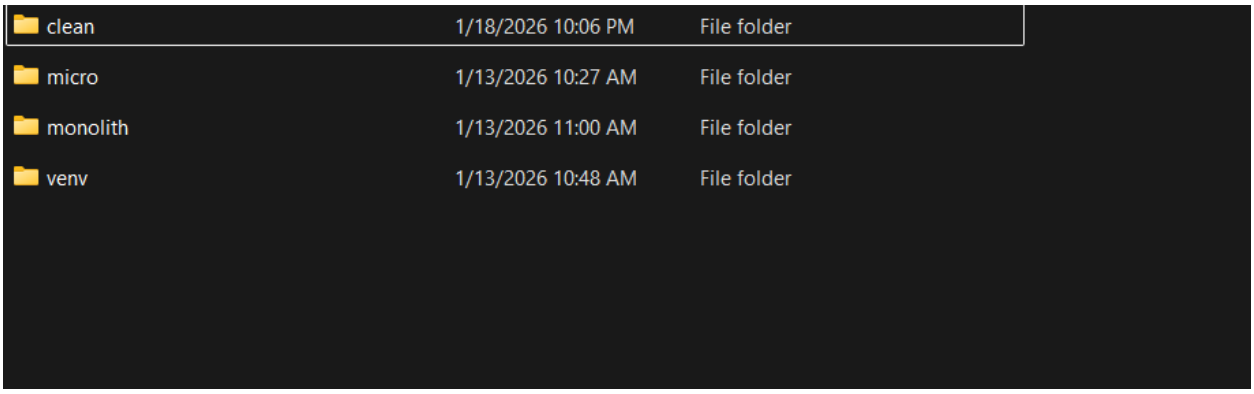
Trần Anh Đăng-B22DCCN211

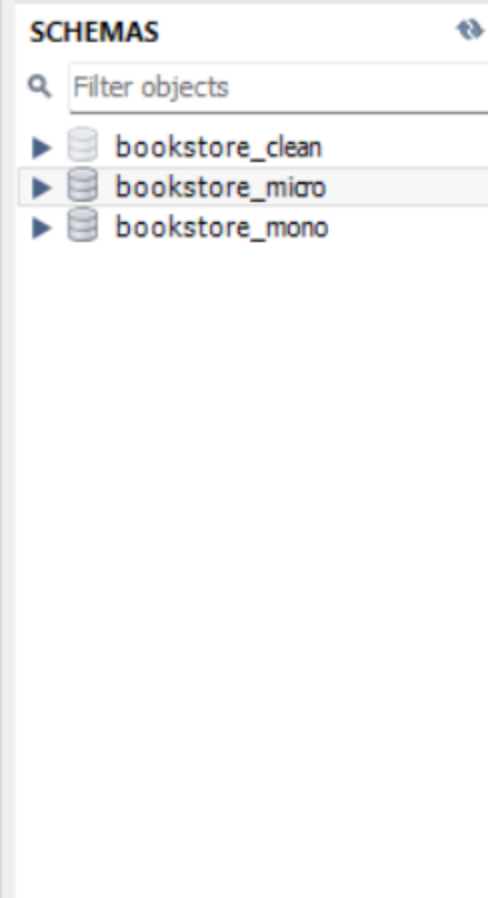
Version A – Monolithic Django

PHẦN 1: TẠO THƯ MỤC & CÀI ĐẶT MÔI TRƯỜNG



Bước 2: Bước 2: Tạo môi trường ảo (venv)

PHẦN 2: CHUẨN BỊ DATABASE (MySQL)



PHẦN 3: TRIỂN KHAI VERSION A (MONOLITHIC)

Bước 1: Tạo Project và App

Bước 2: Cấu hình

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'rest_framework',
    'accounts',
    'books',
    'cart',
]
```

```python
DATABASES = {
    'default': {
   'ENGINE': 'django.db.backends.mysql',
        'NAME': 'bookstore_mono',  # Tên DB đã tạo lúc nãy
        'USER': 'root',
        'PASSWORD': '1234',            # <-- Điền pass MySQL vào đây
        'HOST': 'localhost',
        'PORT': '3306',
    }
}
```

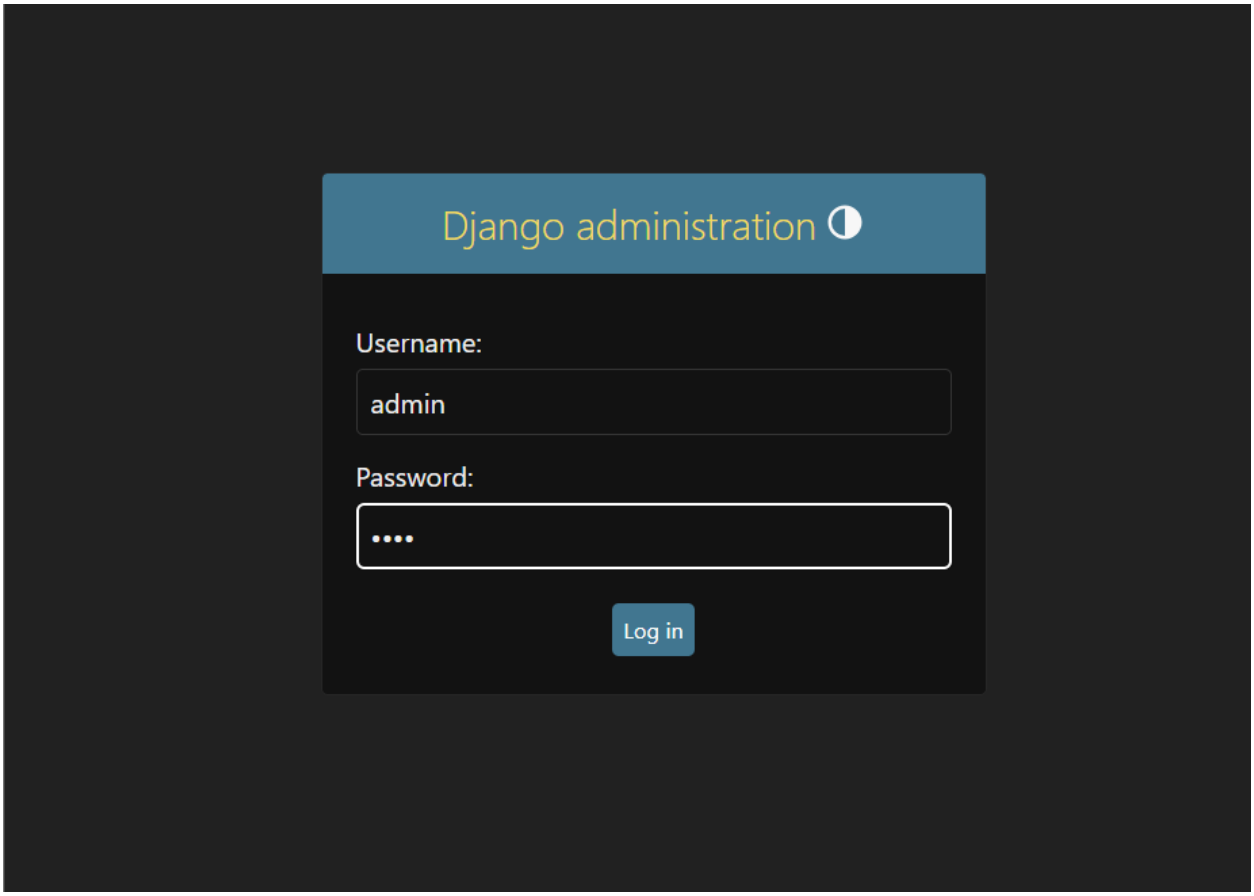
```python
DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
AUTH_USER_MODEL = 'accounts.Customer'
```

Bước 3: Viết Code cho Models

```python
from django.db import models
from django.contrib.auth.models import AbstractUser

class Customer(AbstractUser):
    # Dùng kế thừa AbstractUser là đủ yêu cầu (có sẵn username, pass, email)
    pass
```

```python
from django.db import models

class Book(models.Model):
    title = models.CharField(max_length=255)
    author = models.CharField(max_length=255)
    price = models.DecimalField(max_digits=10, decimal_places=2)
    stock = models.IntegerField(default=0)

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

```python
from django.db import models
from django.conf import settings
from books.models import Book

class Cart(models.Model):
    customer = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE)
    created_at = models.DateTimeField(auto_now_add=True)

class CartItem(models.Model):
    cart = models.ForeignKey(Cart, related_name='items', on_delete=models.CASCADE)
    book = models.ForeignKey(Book, on_delete=models.CASCADE)
    quantity = models.PositiveIntegerField(default=1)
```



BƯỚC 4: Hiện bảng lên trang Admin

```python
from django.contrib import admin
from .models import Book

admin.site.register(Book)
```

```python
from django.contrib import admin
from .models import Cart, CartItem

admin.site.register(Cart)
admin.site.register(CartItem)
```

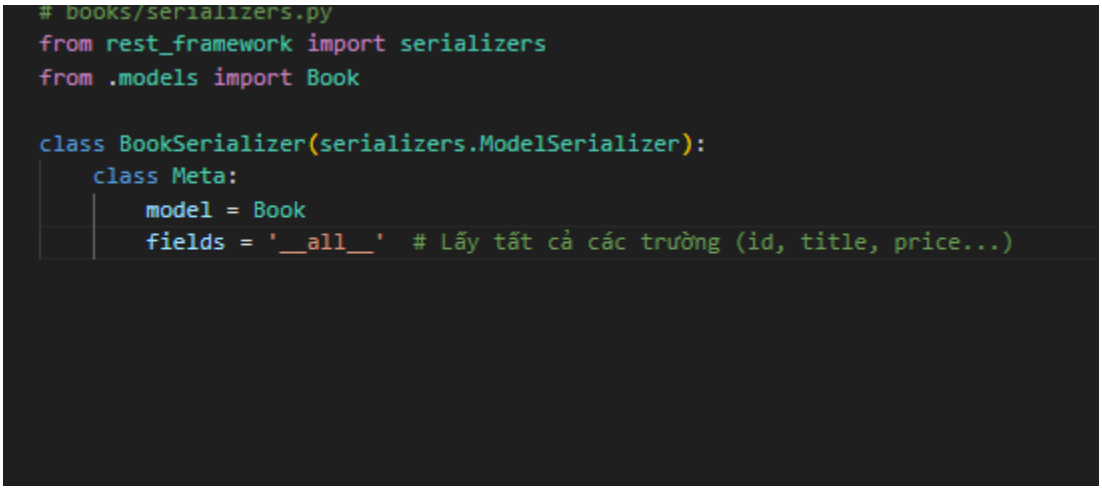
```python
from django.contrib import admin
from django.contrib.auth.admin import UserAdmin
from .models import Customer

admin.site.register(Customer, UserAdmin)
```

Phần 4:Hiển thị danh sách API

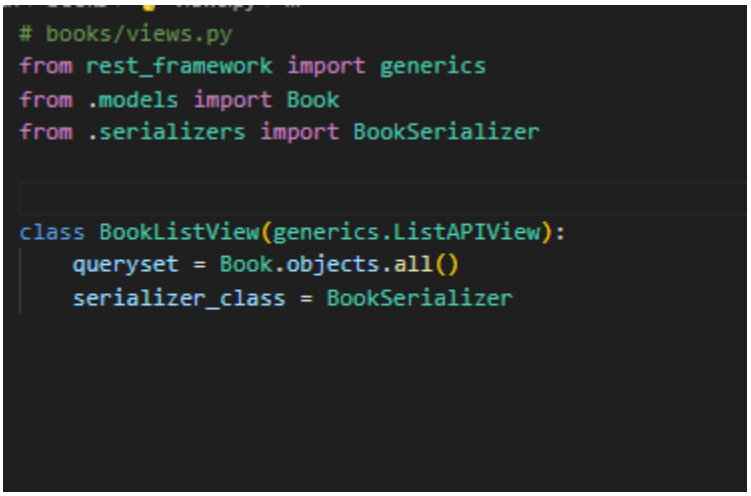
1.List Book

BƯỚC 1: Tạo "Máy dịch" (Serializer)

Vì dữ liệu trong Database là dạng Python Object, muốn trả về cho web/app xem thì phải dịch sang dạng JSON. File này làm nhiệm vụ đó.



BƯỚC 2: Viết Logic

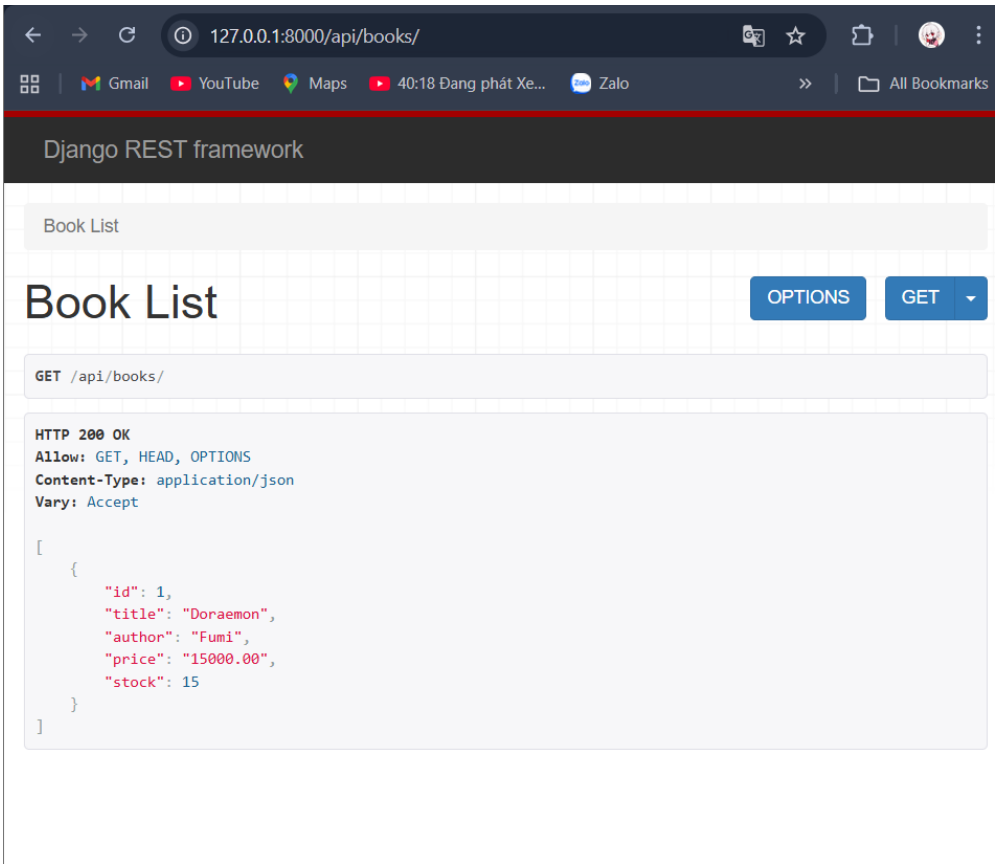


BƯỚC 3: Tạo đường dẫn (URL)

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

urlpatterns = [
    path('api/books/', BookListView.as_view(), name='book-list'),
]
```

```python
# config/urls.py
from django.contrib import admin
from django.urls import path, include # <-- Nhớ thêm include ở đây

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('books.urls')), # <-- Thêm dòng này để nối dây sang app books
]
```

Bước 4:Test API



2.Accounts

BƯỚC 1: Tạo bộ dịch mã (Serializer)

```python
# accounts/serializers.py
from rest_framework import serializers
from django.contrib.auth import get_user_model

Customer = get_user_model()

class CustomerSerializer(serializers.ModelSerializer):
    password = serializers.CharField(write_only=True)

    class Meta:
        model = Customer
        fields = ('id', 'username', 'email', 'password')

    def create(self, validated_data):
        # Hàm này để tạo user và tự động mã hóa password
        user = Customer.objects.create_user(
            username=validated_data['username'],
            email=validated_data.get('email', ''),
            password=validated_data['password']
        )
        return user
```

BƯỚC 2: Viết Logic Đăng ký & Đăng nhập (View)

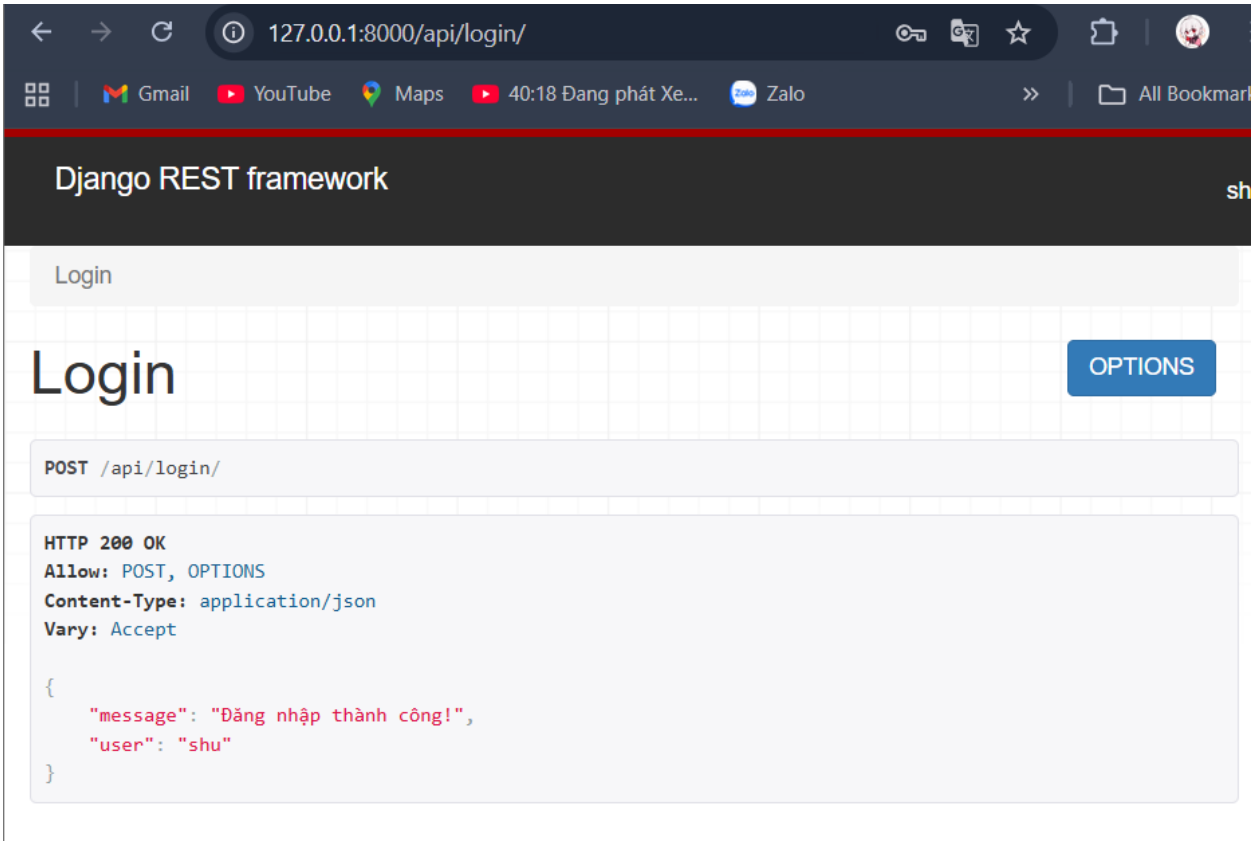
```python
from rest_framework import generics, status, views
from rest_framework.response import Response
from django.contrib.auth import authenticate, login, logout
from .serializers import CustomerSerializer

# 1. API Đăng ký
class RegisterView(generics.CreateAPIView):
    serializer_class = CustomerSerializer

# 2. API Đăng nhập
class LoginView(views.APIView):
    def post(self, request):
        username = request.data.get('username')
        password = request.data.get('password')

        user = authenticate(username=username, password=password)

        if user is not None:
            login(request, user)
            return Response({"message": "Đăng nhập thành công!", "user": username}, status=status.HTTP_200_O
        else:
            return Response({"error": "Sai tài khoản hoặc mật khẩu"}, status=status.HTTP_401_UNAUTHORIZED)

# 3. API Đăng xuất
class LogoutView(views.APIView):
    def post(self, request):
        logout(request)
        return Response({"message": "Đăng xuất thành công!"}, status=status.HTTP_200_OK)
```

BƯỚC 3: Cấu hình đường dẫn (URL)

```python
from .views import RegisterView, LoginView, LogoutView

urlpatterns = [
    path('api/register/', RegisterView.as_view(), name='register'),
    path('api/login/', LoginView.as_view(), name='login'),
    path('api/logout/', LogoutView.as_view(), name='logout'),
]
```

```python
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('books.urls')),    # Đã thêm từ trước
    path('', include('accounts.urls')), # <-- THÊM DÒNG NÀY
]
```

Bước 4:Test api



3.Cart

BƯỚC 1: Tạo bộ dịch mã (Serializer) cho Giỏ hàng

```python
from rest_framework import serializers
from .models import Cart, CartItem
from books.serializers import BookSerializer # Dùng lại cái cũ cho tiện

class CartItemSerializer(serializers.ModelSerializer):
    # Khi xem giỏ hàng sẽ hiện full thông tin sách (Title, Price...)
    book = BookSerializer(read_only=True)
    book_id = serializers.IntegerField(write_only=True)
    class Meta:
        model = CartItem
        fields = ['id', 'book', 'book_id', 'quantity']

class CartSerializer(serializers.ModelSerializer):
    items = CartItemSerializer(many=True, read_only=True)

    class Meta:
        model = Cart
        fields = ['id', 'customer', 'created_at', 'items']
```

BƯỚC 2: Viết Logic Thêm và Xem (View)

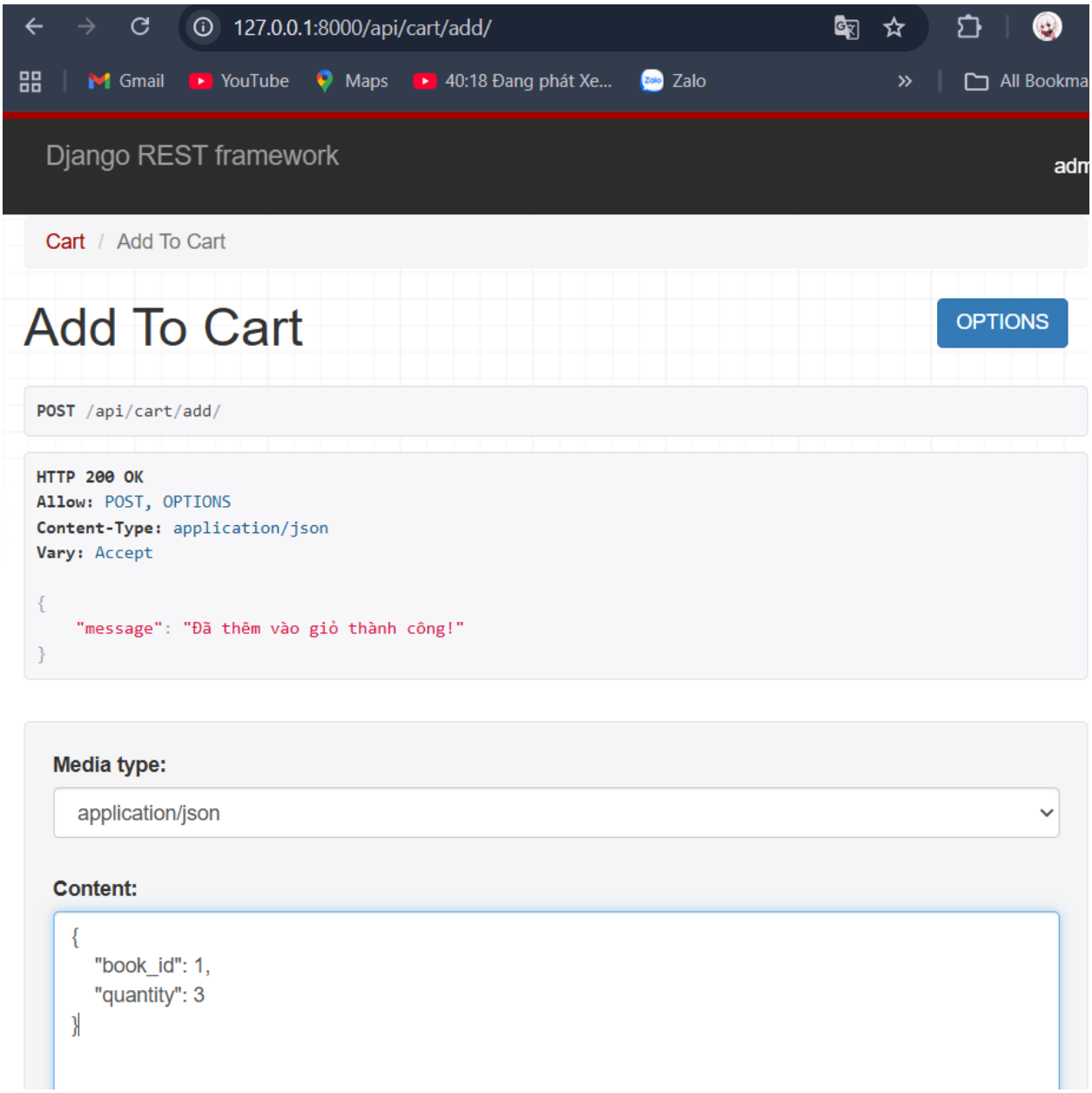
```python
from django.utils.decorators import method_decorator
from django.views.decorators.csrf import csrf_exempt
from rest_framework import generics, status, views, permissions
from rest_framework.response import Response
from .models import Cart, CartItem
from .serializers import CartSerializer
from books.models import Book

class CartView(views.APIView):
    # Chỉ cho phép người đã đăng nhập mới được xem giỏ
    permission_classes = [permissions.IsAuthenticated]

    def get(self, request):
        # Lấy giỏ hàng của user đang đăng nhập
        cart, created = Cart.objects.get_or_create(customer=request.user)
        serializer = CartSerializer(cart)
        return Response(serializer.data)

class AddToCartView(views.APIView):
    permission_classes = [permissions.IsAuthenticated]

    def post(self, request):
        # Lấy dữ liệu gửi lên: book_id và quantity
        book_id = request.data.get('book_id')
        quantity = int(request.data.get('quantity', 1))

        # 1. Tìm hoặc tạo giỏ hàng cho user
        cart, created = Cart.objects.get_or_create(customer=request.user)

        # 2. Kiểm tra sách có tồn tại không
        try:
            book = Book.objects.get(id=book_id)
        except Book.DoesNotExist:
            return Response({"error": "Sách không tồn tại"}, status=status.HTTP_404_NOT_FOUND)

        # 3. Thêm vào giỏ (Nếu có rồi thì cộng dồn, chưa có thì tạo mới)
        cart_item, item_created = CartItem.objects.get_or_create(cart=cart, book=book)

        if not item_created:
            cart_item.quantity += quantity
            cart_item.save()
        else:
            cart_item.quantity = quantity
            cart_item.save()

        return Response({"message": "Đã thêm vào giỏ thành công!"}, status=status.HTTP_200_OK)
```

BƯỚC 3: Nối dây (URL)

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

urlpatterns = [
    path('api/books/', BookListView.as_view(), name='book-list'),
]
```

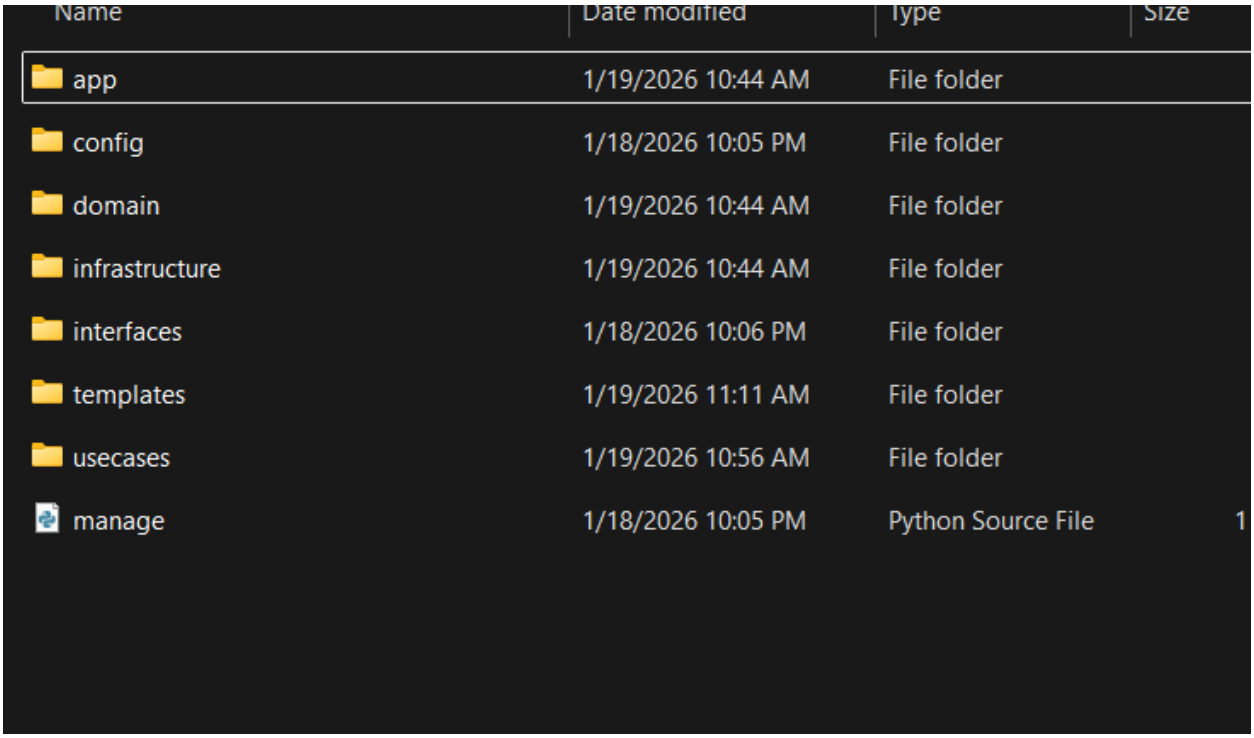
```python
# cart/urls.py
from django.urls import path
from .views import CartView, AddToCartView

urlpatterns = [
    path('api/cart/', CartView.as_view(), name='view-cart'),
    path('api/cart/add/', AddToCartView.as_view(), name='add-to-cart'),
]
```

Bước 4:Test API



PHẦN 2: VERSION B – CLEAN ARCHITECTURE

2. Cấu trúc thư mục (Project Structure)



Dự án được tổ chức theo cấu trúc phân tầng rõ ràng:

- domain/ (Entities): Chứa các thực thể cốt lõi của hệ thống như Customer, Book, Cart. Đây là lớp trong cùng, chứa dữ liệu và logic doanh nghiệp thuần túy nhất, không phụ thuộc vào bất kỳ thư viện bên ngoài nào.
- usecases/ (Application Business Rules): Chứa các logic nghiệp vụ cụ thể của ứng dụng.
    - *Ví dụ:* AddToCartUseCase (Xử lý việc kiểm tra tồn kho, tính toán giá tiền, thêm vào giỏ).
    - Đây là nơi thay thế cho việc viết logic dày đặc trong views.py (Fat Views) của Django truyền thống.
- infrastructure/ (Interface Adapters): Chứa các Repositories.
    - Đóng vai trò cầu nối giữa Use Cases và Database.

    - Use Cases chỉ gọi các hàm trừu tượng (như save, get), còn Infrastructure sẽ thực thi các lệnh Django ORM cụ thể.
- app/ (Frameworks & Drivers): Chứa code đặc thù của Django như views.py, urls.py, serializers.py. Nhiệm vụ duy nhất của View là nhận HTTP Request và chuyển tiếp cho Use Case xử lý.

3. Luồng dữ liệu (Data Flow Implementation)

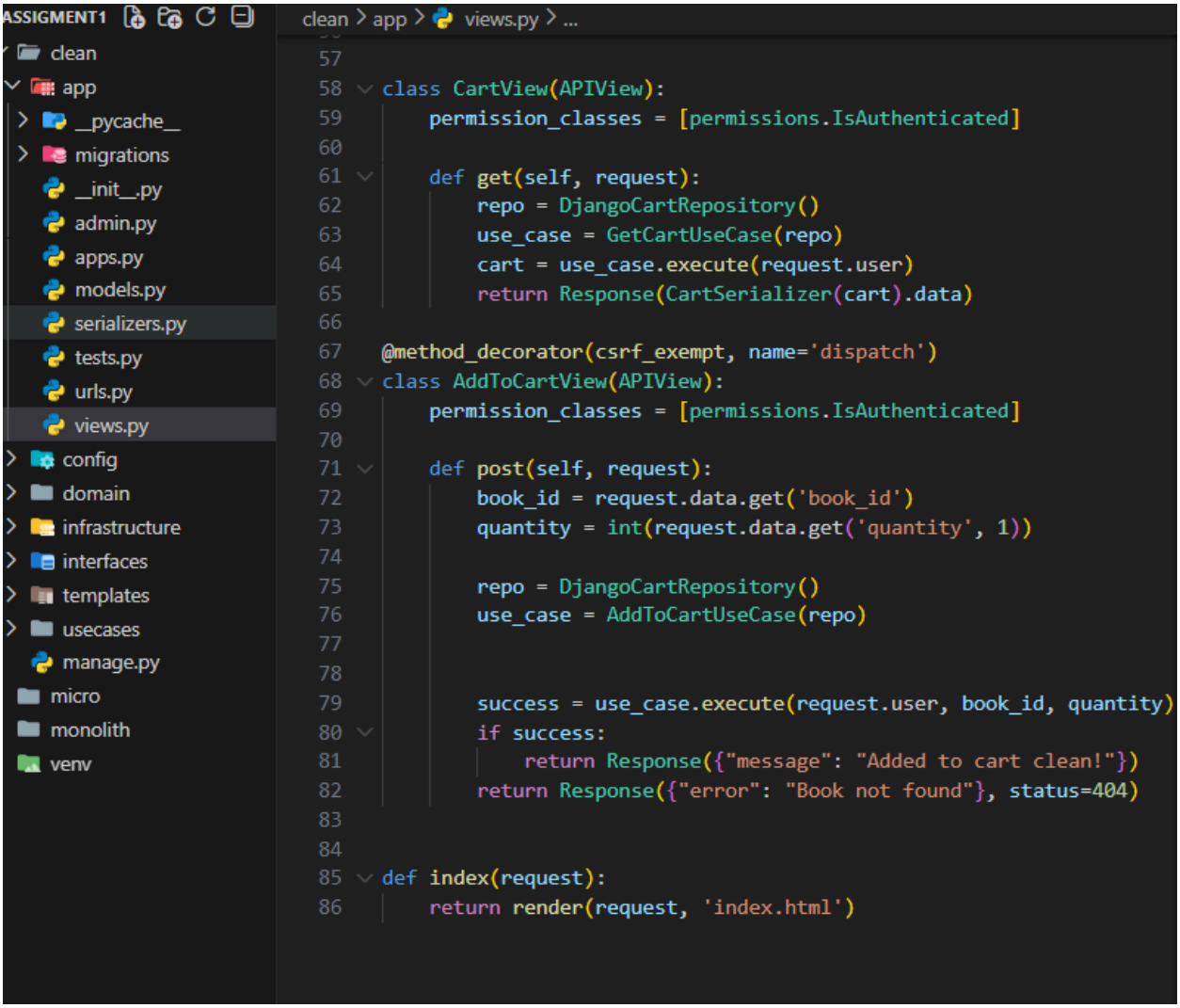
Lấy ví dụ chức năng "Thêm sách vào giỏ hàng" (Add to Cart), luồng dữ liệu di chuyển như sau:

1. Framework Layer (views.py):
    - Nhận POST request từ Client.
    - Khởi tạo DjangoCartRepository (tầng Infrastructure).
    - Khởi tạo AddToCartUseCase và tiêm (inject) repository vào đó.
    - Gọi hàm use_case.execute(user, book_id, quantity).
2. Use Case Layer (usecases/cart.py):
    - Nhận dữ liệu đầu vào.
    - Thực hiện các quy tắc nghiệp vụ (Business Rules).
    - Gọi self.repository.add_to_cart(...) để lưu dữ liệu.
    - *Lưu ý:* Tại đây không có code của Django Views hay HTTP Response.
3. Infrastructure Layer (repositories.py):
    - Thực thi lệnh truy vấn database thực tế thông qua Django ORM (Cart.objects.get_or_create...).
    - Trả kết quả ngược lại cho Use Case.

4. Minh chứng triển khai (Implementation Details)

```
UseCase: Lấy giỏ hàng
ass GetCartUseCase:
  def __init__(self, repository):
      self.repository = repository

  def execute(self, user):
      return self.repository.get_cart(user)

UseCase: Thêm vào giỏ
ass AddToCartUseCase:
  def __init__(self, repository):
      self.repository = repository

  def execute(self, user, book_id, quantity):
      return self.repository.add_to_cart(user, book_id, quantity)
```

```python
class RegisterView(generics.CreateAPIView):
    serializer_class = CustomerSerializer

class LoginView(APIView):
    def post(self, request):
        username = request.data.get('username')
        password = request.data.get('password')
        user = authenticate(username=username, password=password)
        if user:
            login(request, user)
            return Response({"message": "Login Success", "user": use
        return Response({"error": "Invalid credentials"}, status=401

class BookListView(APIView):
    def get(self, request):
        repo = DjangoBookRepository()
        use_case = ListBooksUseCase(repo)
        books = use_case.execute()
        return Response(BookSerializer(books, many=True).data)

@method_decorator(csrf_exempt, name='dispatch')
class AddBookView(APIView):
    permission_classes = [permissions.IsAuthenticated]

    def post(self, request):
        title = request.data.get('title')
        author = request.data.get('author')
        price = request.data.get('price')
        stock = request.data.get('stock')

        repo = DjangoBookRepository()
        use_case = CreateBookUseCase(repo)

        try:
            book = use_case.execute(title, author, price, stock)
            return Response(BookSerializer(book).data, status=status
        except Exception as e:
            return Response({"error": str(e)}, status=status.HTTP_4
```



5. Kết quả Test API

API Đăng kí

Register

OPTIONS

```
GET /api/register/
```

```
HTTP 405 Method Not Allowed
Allow: POST, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "detail": "Method \"GET\" not allowed."
}
```

Raw data | HTML form

Media type:

application/json

Content:

```
{
   "username": "khachhang_clean",
   "email": "clean@example.com",
   "password": "123"
}
```

Register

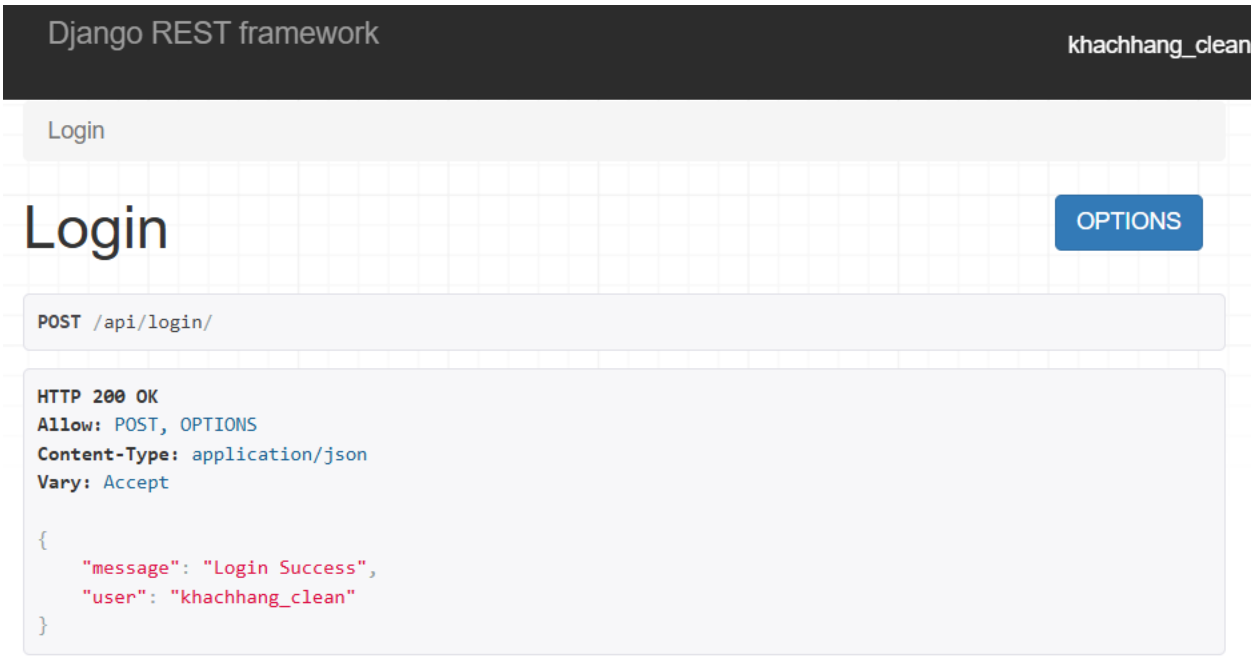
Register

OPTIONS

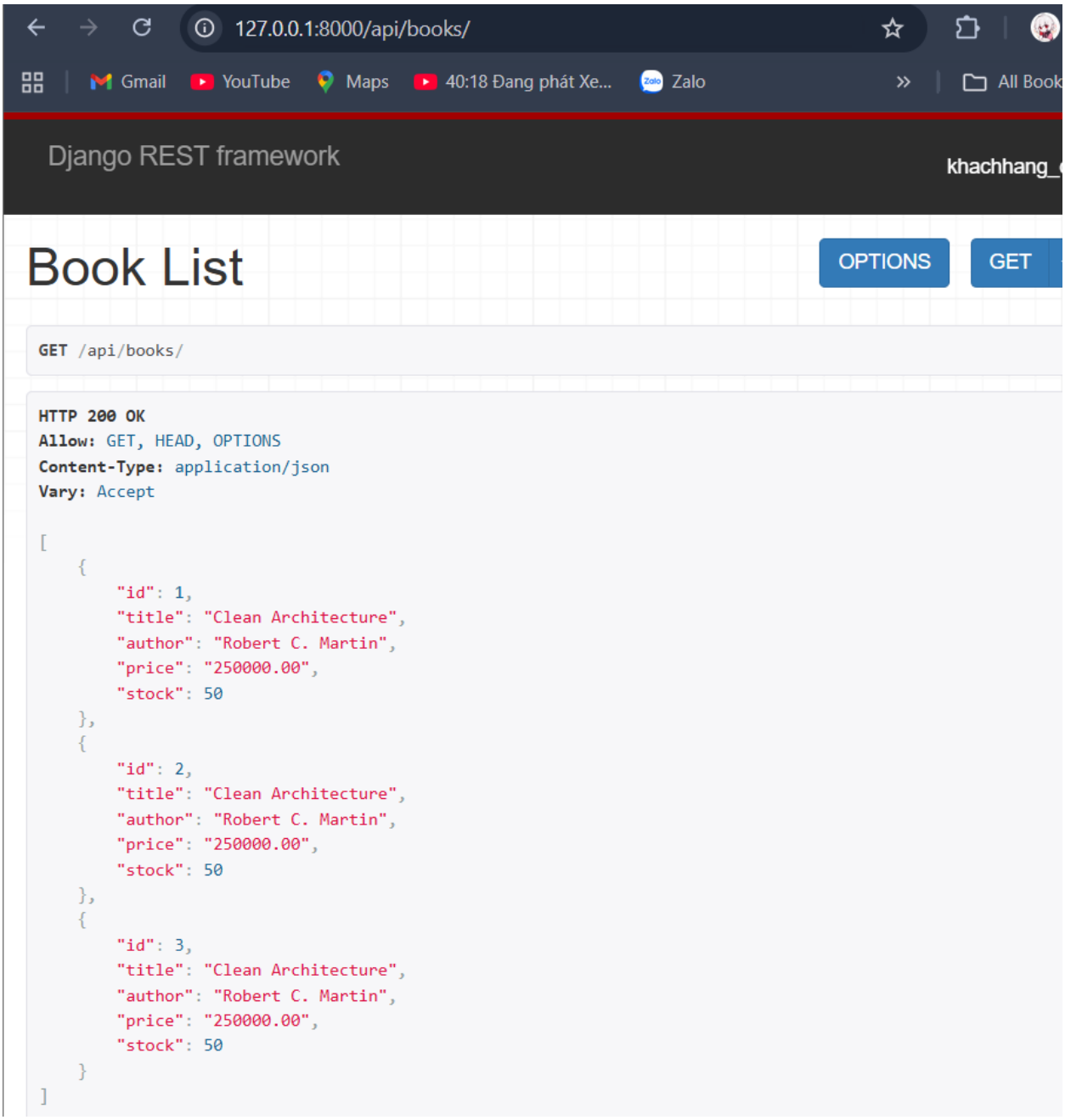
```
POST /api/register/
```

```
HTTP 201 Created
Allow: POST, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "id": 3,
    "username": "khachhang_clean",
    "email": "clean@example.com"
}
```

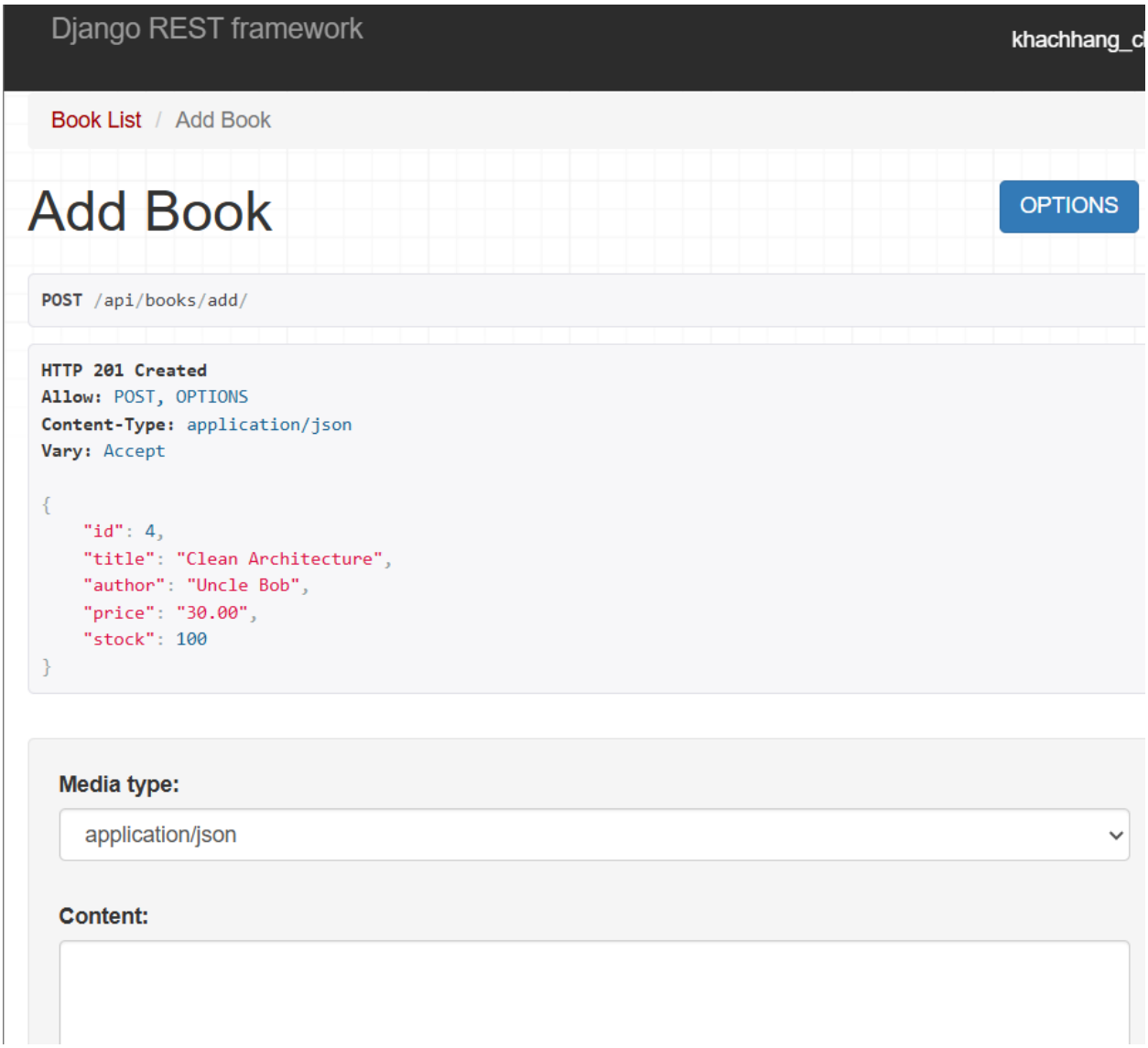
API Đăng nhập



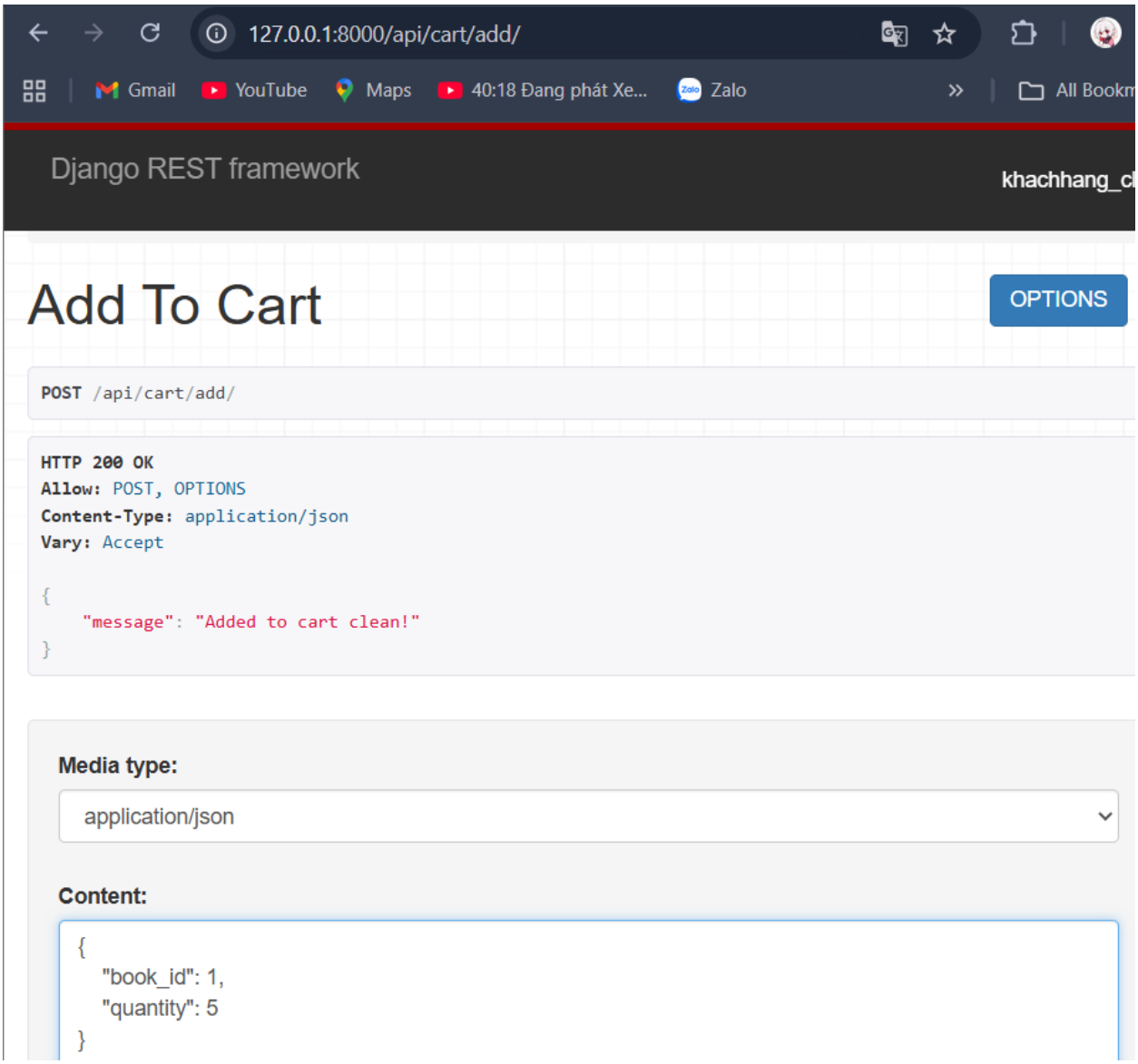
API Xem danh sách sách



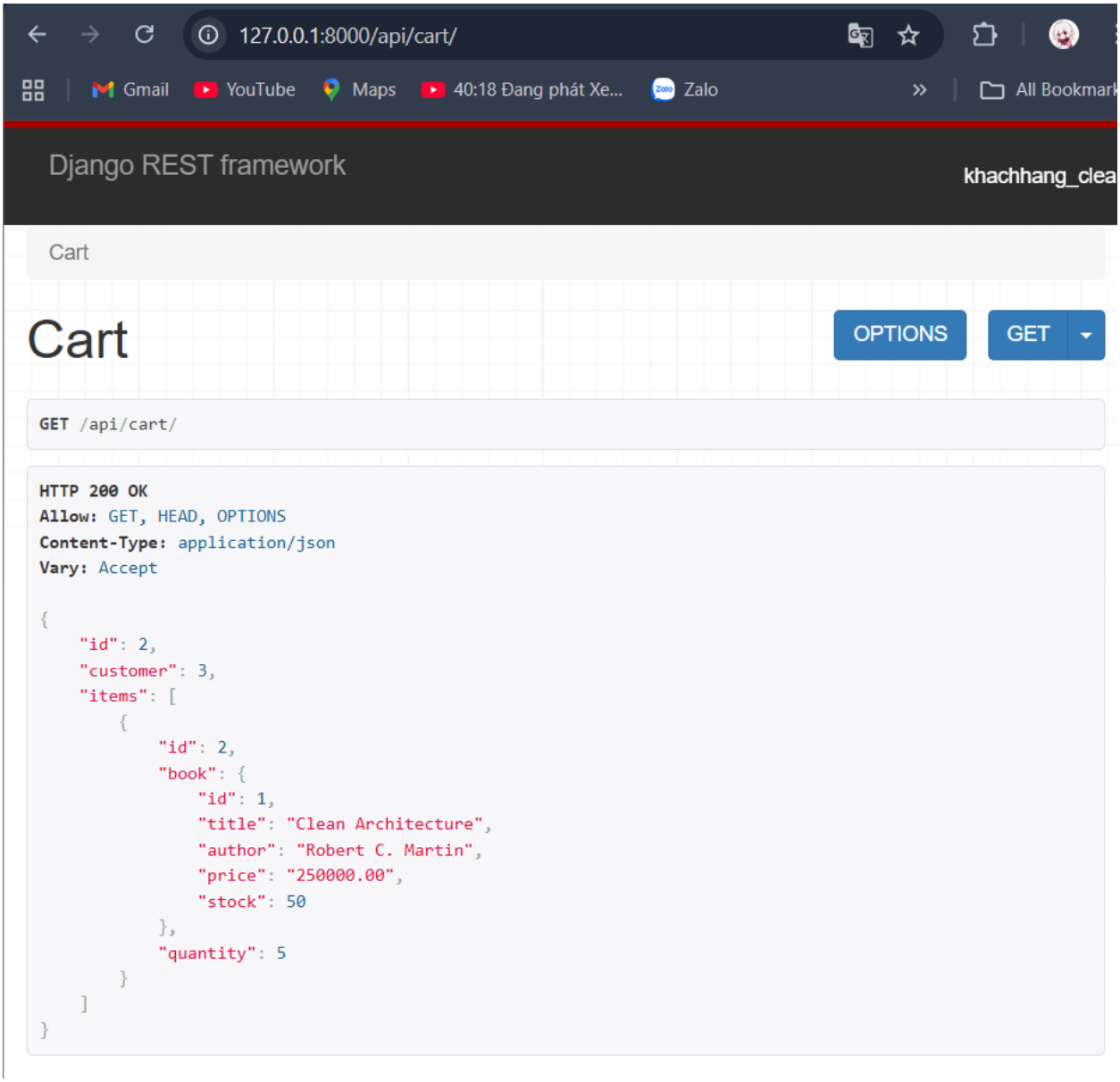
API thêm sách



APT Thêm vào giỏ



API Xem giỏ hàng


127.0.0.1:8000/api/cart/
Gmail
YouTube
Maps
40:18 Đang phát Xe...
Zalo
Django REST framework
khachhang_clea
Cart
Cart
OPTIONS
GET
GET /api/cart/
HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept
{
"id": 2,
"customer": 3,
"items": [
{
"id": 2,
"book": {
"id": 1,
"title": "Clean Architecture",
"author": "Robert C. Martin",
"price": "250000.00",
"stock": 50
},
"quantity": 5
}
]
}

BÁO CÁO PHẦN 3: VERSION C – MICROSERVICES DJANGO

1. Kiến trúc hệ thống (System Architecture)

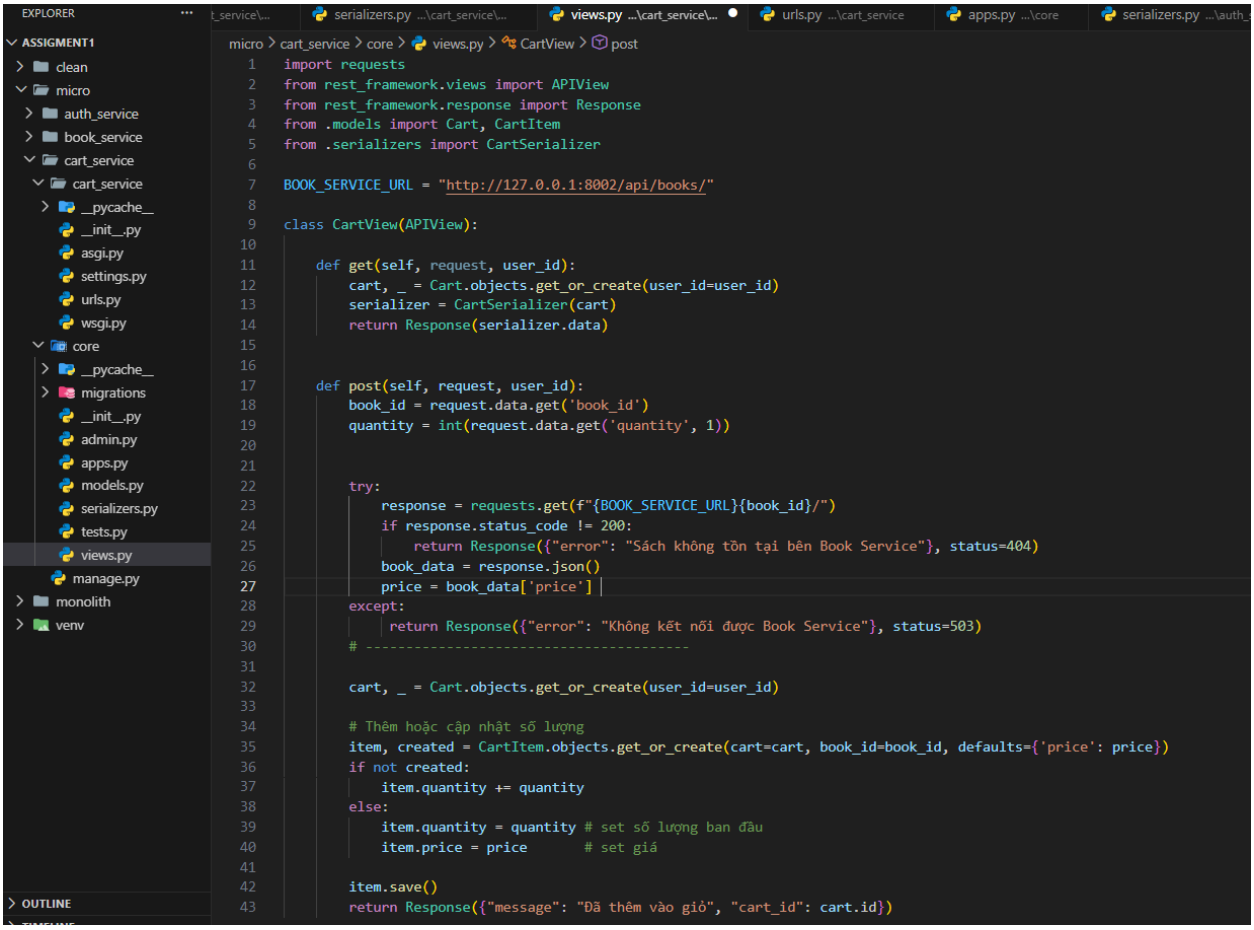
Hệ thống được phân rã thành 3 dịch vụ độc lập (Independent Services), mỗi dịch vụ chạy trên một tiến trình (process) và cổng (port) riêng biệt, giao tiếp với nhau thông qua RESTful APIs.

- Auth Service (Port 8001): Quản lý đăng ký, đăng nhập và thông tin người dùng.
- Book Service (Port 8002): Quản lý kho sách (Catalog), cung cấp thông tin sản phẩm.
- Cart Service (Port 8003): Quản lý giỏ hàng. Dịch vụ này đóng vai trò "Client" gọi sang Book Service để lấy thông tin giá sách trước khi lưu vào giỏ.

2. Minh chứng giao tiếp (Communication Implementation)

Cơ chế: Synchronous HTTP Communication (Giao tiếp đồng bộ qua HTTP).

Thư viện sử dụng: Python requests.



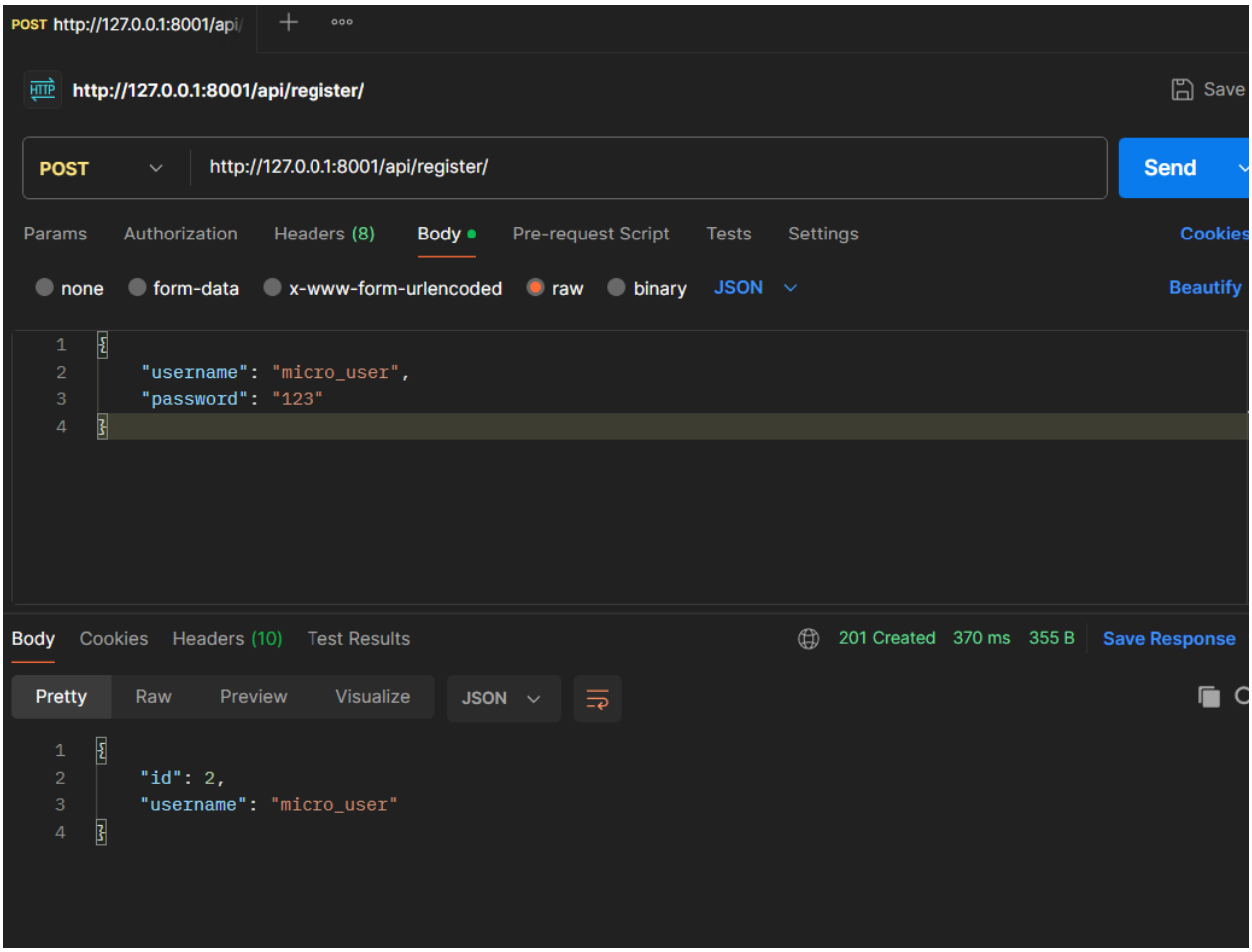
Hình ảnh cho thấy khi User thêm sách vào giỏ, Cart Service thực hiện một GET Request sang Book Service (http://127.0.0.1:8002/...) để xác thực sự tồn tại của sách và lấy giá tiền hiện tại, đảm bảo tính toàn vẹn dữ liệu phân tán.

3. Cấu trúc Database (Database Separation)

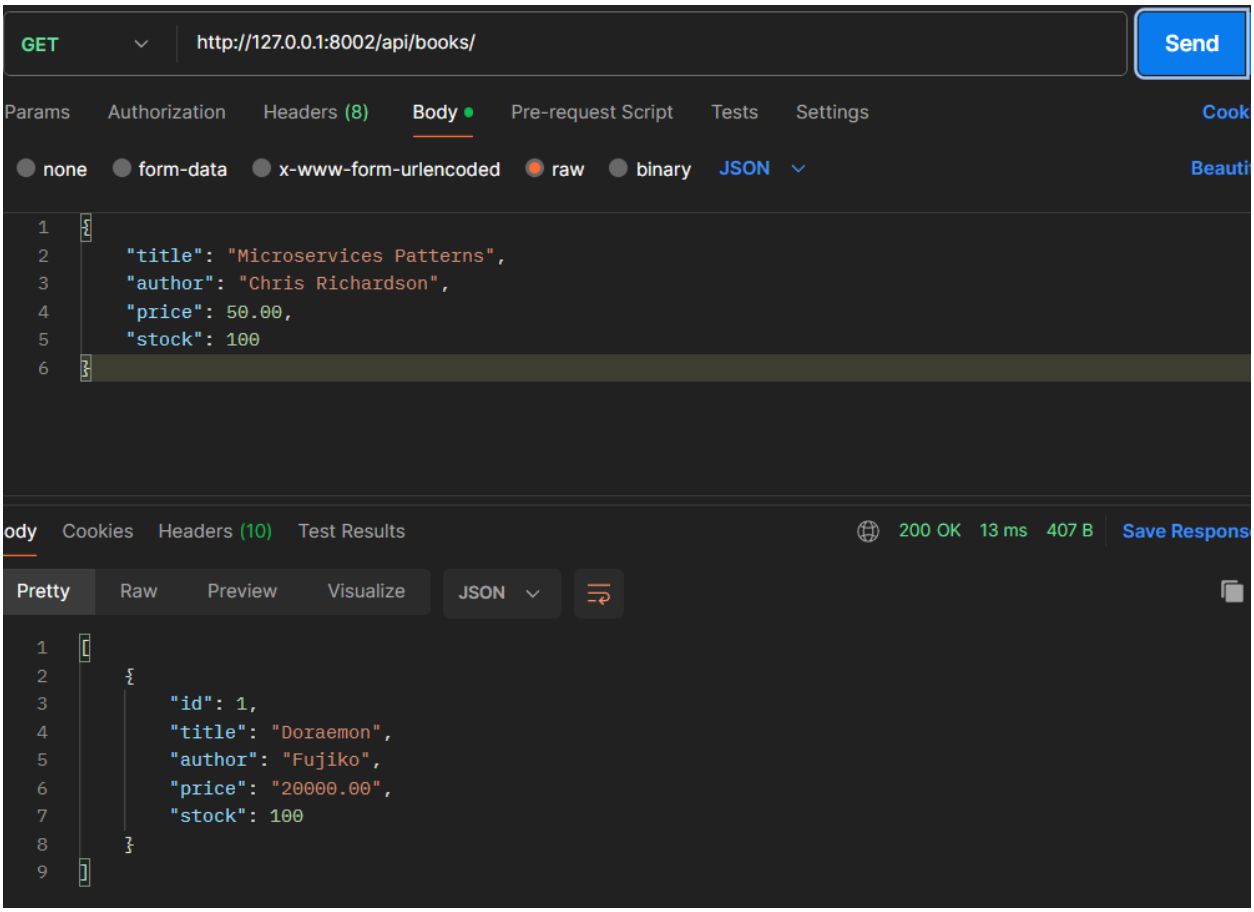
Mặc dù sử dụng chung hệ quản trị MySQL, nhưng các bảng được phân tách logic rõ ràng (Logical Separation):

- auth_...: Các bảng của Auth Service.
- core_book: Bảng của Book Service.
- cart_app_...: Các bảng của Cart Service.

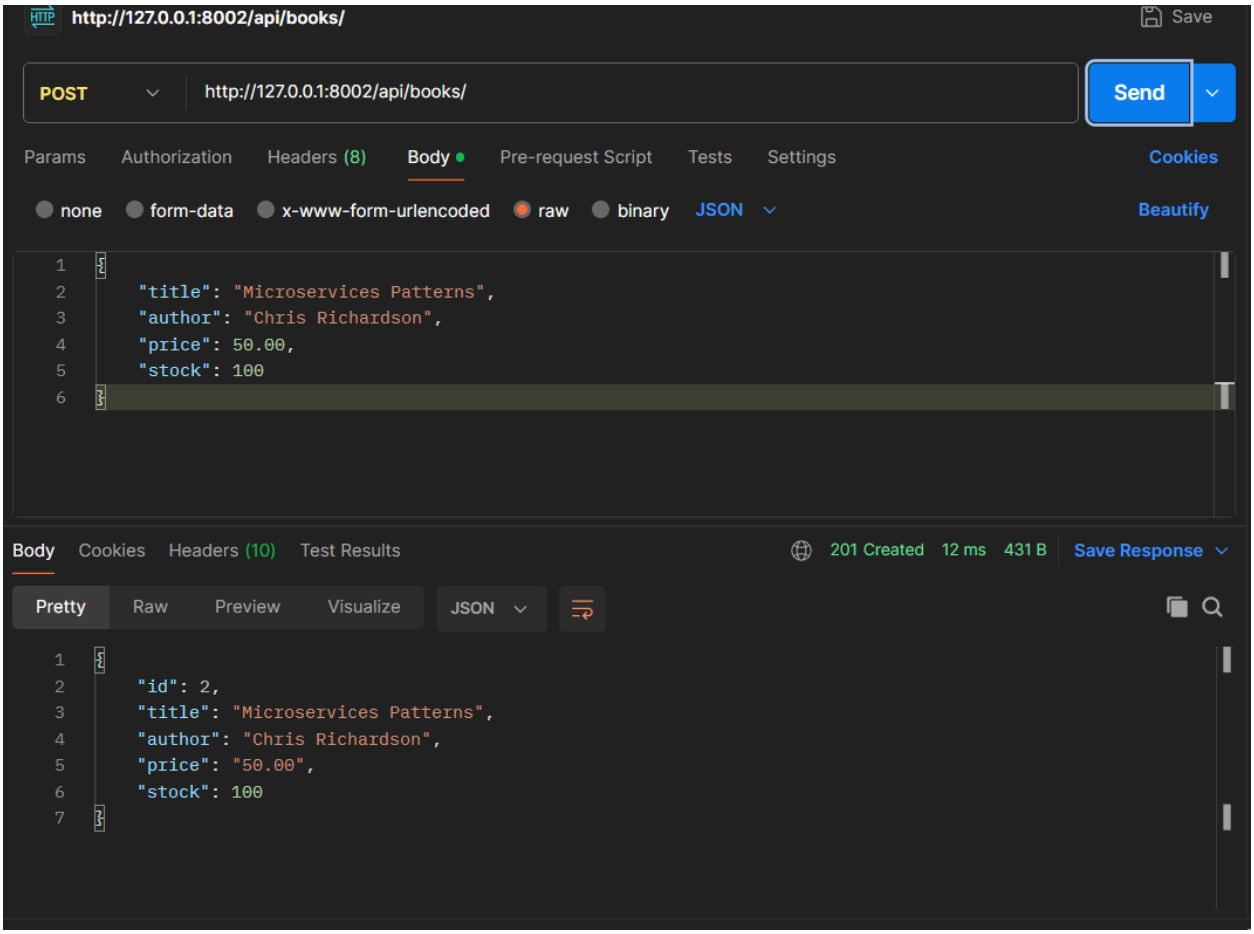
Đăng ký User (Auth Service - Port 8001)



Xem Sách (Book Service - Port 8002)



Tạo sách


http://127.0.0.1:8002/api/books/
Save
POST
http://127.0.0.1:8002/api/books/
Send
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
JSON
Beautify
{
"title": "Microservices Patterns",
"author": "Chris Richardson",
"price": 50.00,
"stock": 100
}
Body
Cookies
Headers (10)
Test Results
201 Created
12 ms
431 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"id": 2,
"title": "Microservices Patterns",
"author": "Chris Richardson",
"price": "50.00",
"stock": 100
}

Thêm vào giỏ hàng