

Trần Anh Đăng-B22DCCN211

Version A – Monolithic Django

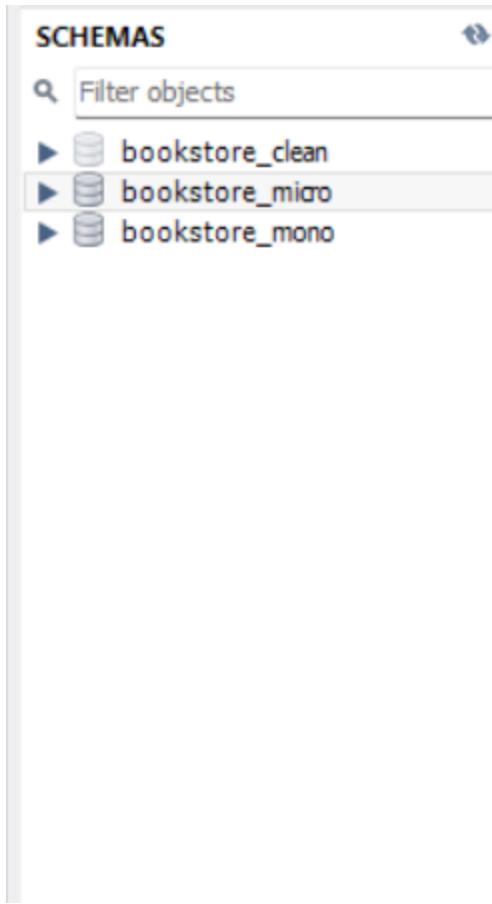
PHẦN 1: TẠO THƯ MỤC & CÀI ĐẶT MÔI TRƯỜNG

📁 clean	1/18/2026 10:06 PM	File folder
📁 micro	1/13/2026 10:27 AM	File folder
📁 monolith	1/13/2026 11:00 AM	File folder
📁 venv	1/13/2026 10:48 AM	File folder

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

```
(venv) D:\django>
```

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',  
]
```

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

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

Bước 3: Viết Code cho Models

```
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

from django.db import models

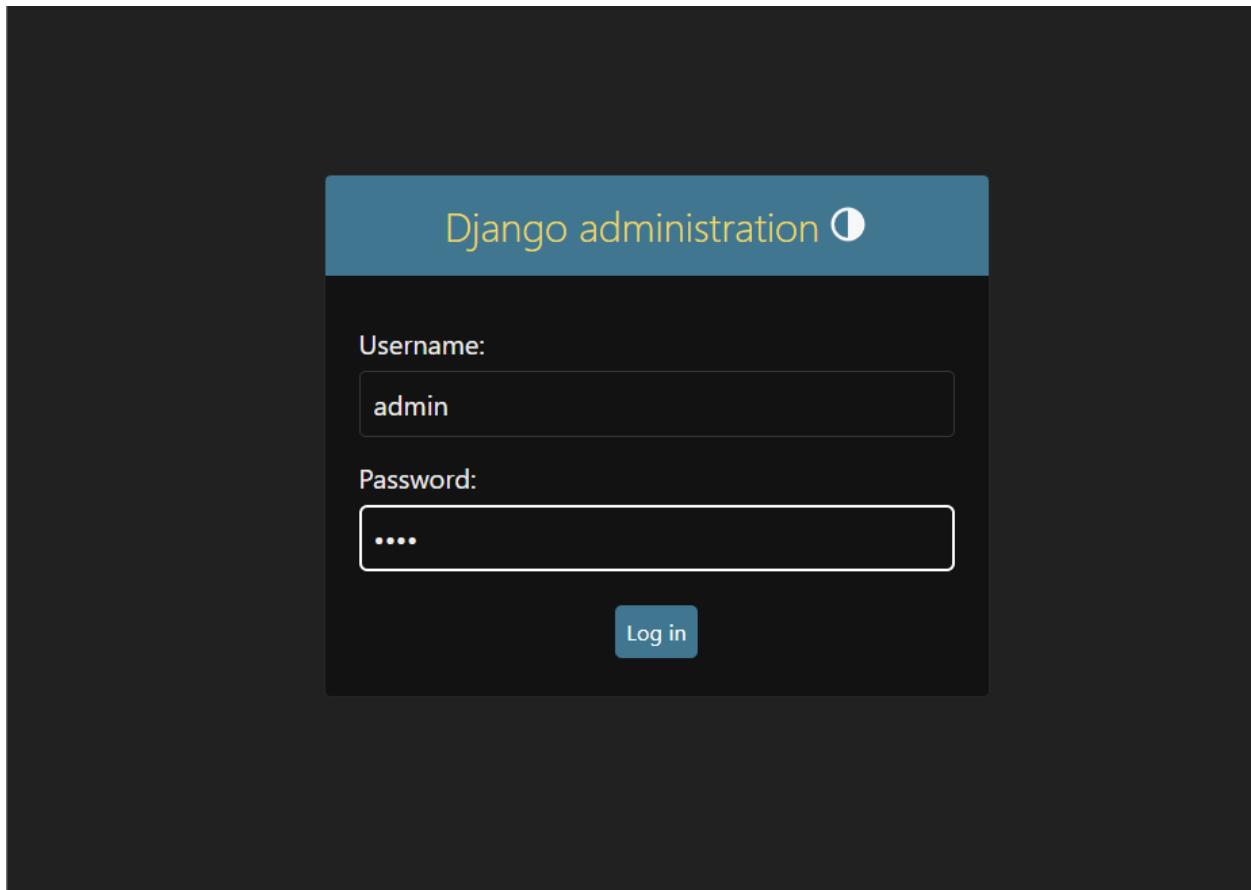
class Book(models.Model):
    title = models.CharField(max_length=255)
    author = models.CharField(max_length=255)
    price = models.DecimalField(max_digits=10, decimal_places=2)
    stock = models.IntegerField(default=0)

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

```
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

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

admin.site.register(Book)
```

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

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

```
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.

```
# books/serializers.py
from rest_framework import serializers
from .models import Book

class BookSerializer(serializers.ModelSerializer):
    class Meta:
        model = Book
        fields = '__all__' # Lấy tất cả các trường (id, title, price...)
```

BƯỚC 2: Viết Logic

```
# books/views.py
from rest_framework import generics
from .models import Book
from .serializers import BookSerializer

class BookListView(generics.ListAPIView):
    queryset = Book.objects.all()
    serializer_class = BookSerializer
```

BƯỚC 3: Tạo đường dẫn (URL)

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

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

```
# 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

The screenshot shows a browser window with the URL `127.0.0.1:8000/api/books/`. The page title is "Django REST framework". Below the title, there's a header "Book List" with two buttons: "OPTIONS" and "GET". A dropdown menu next to "GET" shows "GET /api/books/". The main content area displays the response from a GET request:

```
HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

[{"id": 1, "title": "Doraemon", "author": "Fumi", "price": "15000.00", "stock": 15}]
```

2.Accounts

BUỚC 1: Tạo bộ dịch mã (Serializer)

```
# 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
```

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

```

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_OK)
        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)

```

from django.urls import path
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'),
]

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

Django REST framework

Login

Login

OPTIONS

POST /api/login/

```
HTTP 200 OK
Allow: POST, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "message": "Đăng nhập thành công!",
    "user": "shu"
}
```

3.Cart

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

```
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)

```
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)
```

BUỚC 3: Nối dây (URL)

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

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

```
# 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

The screenshot shows a web browser window with the URL `127.0.0.1:8000/api/cart/add/` in the address bar. The page title is "Django REST framework". The main content area displays an "Add To Cart" section with a "OPTIONS" button. Below it, there's a "POST /api/cart/add/" button. A response block shows the following details:

HTTP 200 OK
Allow: POST, OPTIONS
Content-Type: application/json
Vary: Accept

```
{  
    "message": "Đã thêm vào giỏ thành công!"  
}
```

Below this, there are two input fields: "Media type:" set to "application/json" and "Content:" containing the following JSON payload:

```
{  
    "book_id": 1,  
    "quantity": 3  
}
```

PHẦN 2: VERSION B – CLEAN ARCHITECTURE

2. Cấu trúc thư mục (Project Structure)

Name	Date modified	Type	Size
📁 app	1/19/2026 10:44 AM	File folder	
📁 config	1/18/2026 10:05 PM	File folder	
📁 domain	1/19/2026 10:44 AM	File folder	
📁 infrastructure	1/19/2026 10:44 AM	File folder	
📁 interfaces	1/18/2026 10:06 PM	File folder	
📁 templates	1/19/2026 11:11 AM	File folder	
📁 usecases	1/19/2026 10:56 AM	File folder	
manage.py	1/18/2026 10:05 PM	Python Source File	1

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)
```

```
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": user})
        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.HTTP_201_CREATED)
        except Exception as e:
            return Response({"error": str(e)}, status=status.HTTP_400_BAD_REQUEST)
```

The screenshot shows a code editor with a dark theme. On the left is a file tree for a project named 'ASSIGNMENT1'. The 'app' directory contains several files: __init__.py, admin.py, apps.py, models.py, serializers.py (which is selected), tests.py, urls.py, and views.py. Other parts of the project include config, domain, infrastructure, interfaces, templates, usecases, manage.py, micro, monolith, and venv. The main pane displays Python code for an API view:

```
clean > app > views.py > ...
57
58     class CartView(APIView):
59         permission_classes = [permissions.IsAuthenticated]
60
61     def get(self, request):
62         repo = DjangoCartRepository()
63         use_case = GetCartUseCase(repo)
64         cart = use_case.execute(request.user)
65         return Response(CartSerializer(cart).data)
66
67     @method_decorator(csrf_exempt, name='dispatch')
68     class AddToCartView(APIView):
69         permission_classes = [permissions.IsAuthenticated]
70
71     def post(self, request):
72         book_id = request.data.get('book_id')
73         quantity = int(request.data.get('quantity', 1))
74
75         repo = DjangoCartRepository()
76         use_case = AddToCartUseCase(repo)
77
78
79         success = use_case.execute(request.user, book_id, quantity)
80     if success:
81         return Response({"message": "Added to cart clean!"})
82     return Response({"error": "Book not found"}, status=404)
83
84
85     def index(request):
86         return render(request, 'index.html')
```

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

Django REST framework khachhang_clean

Login

Login

OPTIONS

POST /api/login/

```
HTTP 200 OK
Allow: POST, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "message": "Login Success",
    "user": "khachhang_clean"
}
```

API Xem danh sách sách

The screenshot shows a browser window with the URL `127.0.0.1:8000/api/books/`. The page title is "Django REST framework" and the user is identified as "khachhang_". Below the title, there's a heading "Book List" with two buttons: "OPTIONS" and "GET". A code block displays the response for a GET request to `/api/books/`, showing the HTTP headers and a JSON array of three book objects.

```
HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

[
    {
        "id": 1,
        "title": "Clean Architecture",
        "author": "Robert C. Martin",
        "price": "250000.00",
        "stock": 50
    },
    {
        "id": 2,
        "title": "Clean Architecture",
        "author": "Robert C. Martin",
        "price": "250000.00",
        "stock": 50
    },
    {
        "id": 3,
        "title": "Clean Architecture",
        "author": "Robert C. Martin",
        "price": "250000.00",
        "stock": 50
    }
]
```

API thêm sách

Django REST framework

khachhang_c

Book List / Add Book

Add Book

OPTIONS

POST /api/books/add/

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

```
{  
    "id": 4,  
    "title": "Clean Architecture",  
    "author": "Uncle Bob",  
    "price": "30.00",  
    "stock": 100  
}
```

Media type:
application/json

Content:

APT Thêm vào giỏ

The screenshot shows a web browser window with the URL `127.0.0.1:8000/api/cart/add/`. The page title is "Django REST framework". On the right, there is a user profile with the name "khachhang_c". A large button labeled "Add To Cart" is centered on the page. To its right is a blue "OPTIONS" button. Below these buttons, there is a code block containing a POST request example:

```
POST /api/cart/add/  
  
HTTP 200 OK  
Allow: POST, OPTIONS  
Content-Type: application/json  
Vary: Accept  
  
{  
    "message": "Added to cart clean!"  
}
```

Below the code block, there are two input fields. The first field is labeled "Media type:" and contains "application/json". The second field is labeled "Content:" and contains a JSON object:

```
{  
    "book_id": 1,  
    "quantity": 5  
}
```

API Xem giỏ hàng

← → ⌂ ① 127.0.0.1:8000/api/cart/ ⌂ ☆ ⌂ ⌂ ⌂ ⌂ ⌂ ⌂

Gmail YouTube Maps 4:18 Đang phát Xe... Zalo ⌂ All Bookmarks

Django REST framework khachhang_clean

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.

```

micro > cart_service > core > views.py > CartView > post
1 import requests
2 from rest_framework.views import APIView
3 from rest_framework.response import Response
4 from .models import Cart, CartItem
5 from .serializers import CartSerializer
6
7 BOOK_SERVICE_URL = "http://127.0.0.1:8002/api/books/"
8
9 class CartView(APIView):
10
11     def get(self, request, user_id):
12         cart, _ = Cart.objects.get_or_create(user_id=user_id)
13         serializer = CartSerializer(cart)
14         return Response(serializer.data)
15
16
17     def post(self, request, user_id):
18         book_id = request.data.get('book_id')
19         quantity = int(request.data.get('quantity', 1))
20
21
22         try:
23             response = requests.get(f"{BOOK_SERVICE_URL}{book_id}/")
24             if response.status_code != 200:
25                 return Response({"error": "Sách không tồn tại bên Book Service"}, status=404)
26             book_data = response.json()
27             price = book_data['price']
28         except:
29             return Response({"error": "Không kết nối được Book Service"}, status=503)
30
31
32         cart, _ = Cart.objects.get_or_create(user_id=user_id)
33
34         # Thêm hoặc cập nhật số lượng
35         item, created = CartItem.objects.get_or_create(cart=cart, book_id=book_id, defaults={'price': price})
36         if not created:
37             item.quantity += quantity
38         else:
39             item.quantity = quantity # set số lượng ban đầu
40             item.price = price # set giá
41
42         item.save()
43         return Response({"message": "Đã thêm vào giỏ", "cart_id": cart.id})

```

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/...](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)

The screenshot shows the Postman application interface. At the top, there is a header bar with the URL `http://127.0.0.1:8001/api/`. Below the header, the main window displays a request configuration for a `POST` method to `http://127.0.0.1:8001/api/register/`. The `Body` tab is selected, showing a JSON payload:

```
1 {  
2     "username": "micro_user",  
3     "password": "123"  
4 }
```

Below the request configuration, the response section is visible. It shows a `201 Created` status with a response time of `370 ms` and a response size of `355 B`. The `Body` tab is selected, displaying the created user object:

```
1 {  
2     "id": 2,  
3     "username": "micro_user"  
4 }
```

Xem Sách (Book Service - Port 8002)

The screenshot shows the Postman application interface. At the top, a header bar displays "GET" and the URL "http://127.0.0.1:8002/api/books/". To the right is a large blue "Send" button. Below the header are tabs for "Params", "Authorization", "Headers (8)", "Body" (which is currently selected), "Pre-request Script", "Tests", and "Settings". Under the "Body" tab, there are radio buttons for "none", "form-data", "x-www-form-urlencoded", "raw" (which is selected), and "binary". A dropdown menu next to "raw" includes "JSON". On the far right, there are "Cookies" and "Beautiful" buttons.

The main content area shows a JSON object with the following code:

```
1 [ {  
2     "title": "Microservices Patterns",  
3     "author": "Chris Richardson",  
4     "price": 50.00,  
5     "stock": 100  
6 } ]
```

Below this, a status bar shows "Body" and "Cookies" tabs, followed by "Headers (10)" and "Test Results". On the right side of the status bar are icons for "Save Response" and a clipboard. The status bar also displays "200 OK 13 ms 407 B".

The "Body" section contains tabs for "Pretty" (selected), "Raw", "Preview", "Visualize", and "JSON" (with a dropdown arrow). The "Pretty" view shows the same JSON object as above. The "Raw" view shows the raw JSON string:

```
[ {  
    "id": 1,  
    "title": "Doraemon",  
    "author": "Fujiko",  
    "price": "20000.00",  
    "stock": 100  
}]
```

Tạo sách

The screenshot shows the Postman application interface. At the top, the URL is set to `http://127.0.0.1:8002/api/books/`. The method is selected as `POST`. The `Body` tab is active, showing the following JSON payload:

```
1
2   "title": "Microservices Patterns",
3   "author": "Chris Richardson",
4   "price": 50.00,
5   "stock": 100
6 }
```

Below the body, the response details are shown: `201 Created`, `12 ms`, and `431 B`. The response body is also displayed in pretty JSON format:

```
1
2   "id": 2,
3   "title": "Microservices Patterns",
4   "author": "Chris Richardson",
5   "price": "50.00",
6   "stock": 100
7 }
```

Thêm vào giỏ hàng

The screenshot shows the Postman application interface. At the top, there is a header bar with the URL `http://127.0.0.1:8003/api/cart/1/`. Below the header, the main interface has a "POST" method selected and the same URL. A large blue "Send" button is visible on the right.

The "Body" tab is active, showing a JSON payload:

```
1 [ {  
2   "book_id": 1,  
3   "quantity": 2  
4 } ]
```

Below the body, the response section shows a status of 200 OK with a response time of 38 ms and a size of 376 B. The "Pretty" tab is selected in the response view, displaying the following JSON:

```
1 [ {  
2   "message": "Đã thêm vào giỏ",  
3   "cart_id": 1  
4 } ]
```