

# Trần Anh Đăng-B22DCCN211

Version A – Monolithic Django

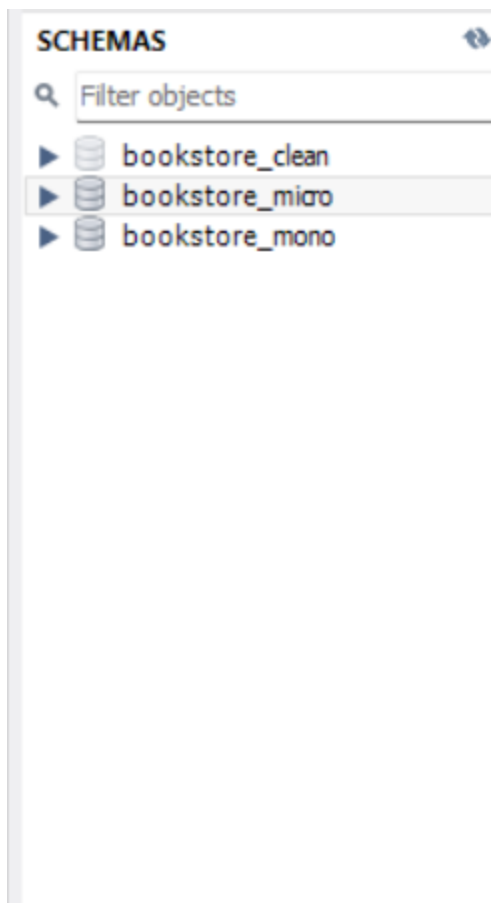
## PHẦN 1: TẠO THƯ MỤC & CÀI ĐẶT MÔI TRƯỜNG

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)
```

## Django administration

Username:

Password:

## 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)

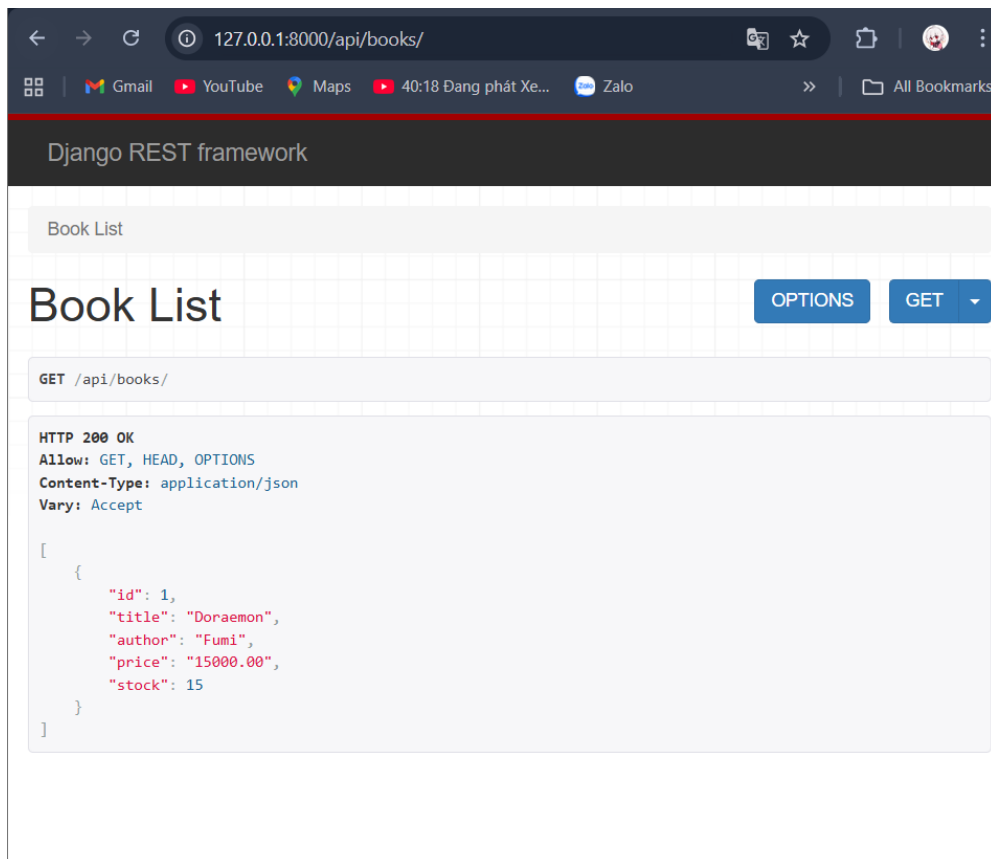
```
from django.urls import path
from .views import BookListView

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 web browser at the address `127.0.0.1:8000/api/books/`. The page displays the Django REST framework interface for the 'Book List' endpoint. The endpoint is `GET /api/books/`. The response is an HTTP 200 OK with the following headers: `Allow: GET, HEAD, OPTIONS`, `Content-Type: application/json`, and `Vary: Accept`. The response body is a JSON array containing one book object:

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

## 2.Accounts

### BƯỚC 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
```

### BƯỚC 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)

```

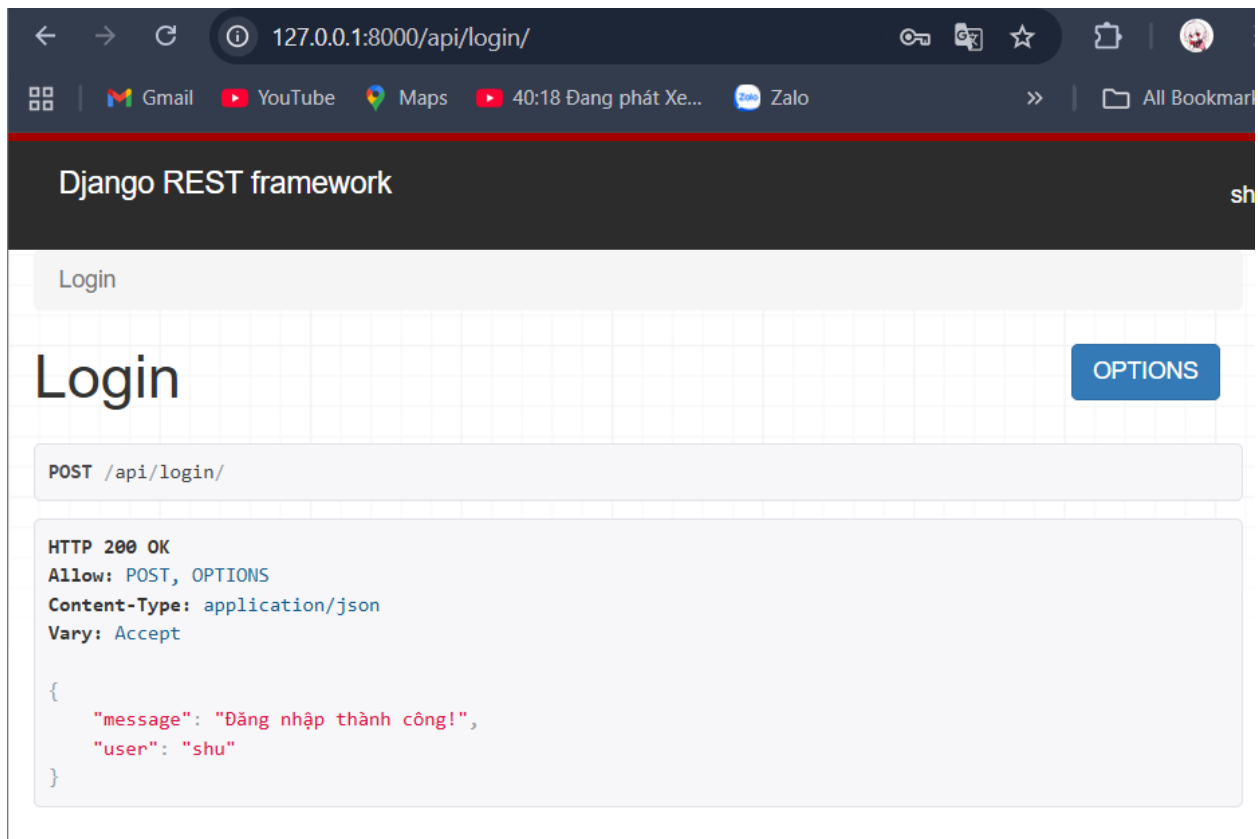
from django.urls import path
from .views import RegisterView, LoginView, LogoutView

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



### 3.Cart

#### BƯỚC 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)
```

### BƯỚC 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 at the URL `127.0.0.1:8000/api/cart/add/`. The browser's address bar and tabs are visible at the top. The page title is "Django REST framework". Below the title, there is a breadcrumb trail: `Cart / Add To Cart`. The main heading is "Add To Cart", and there is a blue button labeled "OPTIONS".

The API details section shows the following information:

- Method:** `POST` `/api/cart/add/`
- Status:** `HTTP 200 OK`
- Allow:** `POST, OPTIONS`
- Content-Type:** `application/json`
- Vary:** `Accept`

The response body is displayed as a JSON object:

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

Below the response, there is a section for "Media type:" with a dropdown menu showing `application/json`. Underneath, the "Content:" section shows the request body as a JSON object:

```
{  "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	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
class GetCartUseCase:
    def __init__(self, repository):
        self.repository = repository

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

UseCase: Thêm vào giỏ
class 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)

```

```
ASSIGNMENT1 clean > app > views.py > ...
clean
├── app
│   ├── __pycache__
│   ├── migrations
│   ├── __init__.py
│   ├── admin.py
│   ├── apps.py
│   ├── models.py
│   ├── serializers.py
│   ├── tests.py
│   ├── urls.py
│   └── views.py
├── config
├── domain
├── infrastructure
├── interfaces
├── templates
├── usecases
├── manage.py
├── micro
├── monolith
└── venv
```

```
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         success = use_case.execute(request.user, book_id, quantity)
79         if success:
80             return Response({"message": "Added to cart clean!"})
81         return Response({"error": "Book not found"}, status=404)
82
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

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 web browser at the URL `127.0.0.1:8000/api/books/`. The browser's address bar and tabs are visible at the top. The page header includes the text "Django REST framework" and a user profile icon labeled "khachhang\_". The main content area is titled "Book List" and features two buttons: "OPTIONS" and "GET". Below the title, a light gray box displays the HTTP response details for a GET request to `/api/books/`. The response status is "HTTP 200 OK", and the allowed methods are "GET, HEAD, OPTIONS". The "Content-Type" is "application/json" and "Vary" is "Accept". The response body is a JSON array containing three book objects, each with the same data: id, title, author, price, and stock.

127.0.0.1:8000/api/books/

Gmail YouTube Maps 40:18 Đang phát Xe... Zalo All Book

Django REST framework khachhang\_

# Book List

OPTIONS GET

GET /api/books/

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

Django REST framework khachhang\_c

# Add To Cart OPTIONS

POST /api/cart/add/

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

```
{  
  "message": "Added to cart clean!"  
}
```

Media type:  
application/json

Content:  
{  
 "book\_id": 1,  
 "quantity": 5  
}

## API Xem giỏ hàng

127.0.0.1:8000/api/cart/

Gmail

YouTube

Maps

40:18 Đang phát Xe...

Zalo

>>

All Bookmarks

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)

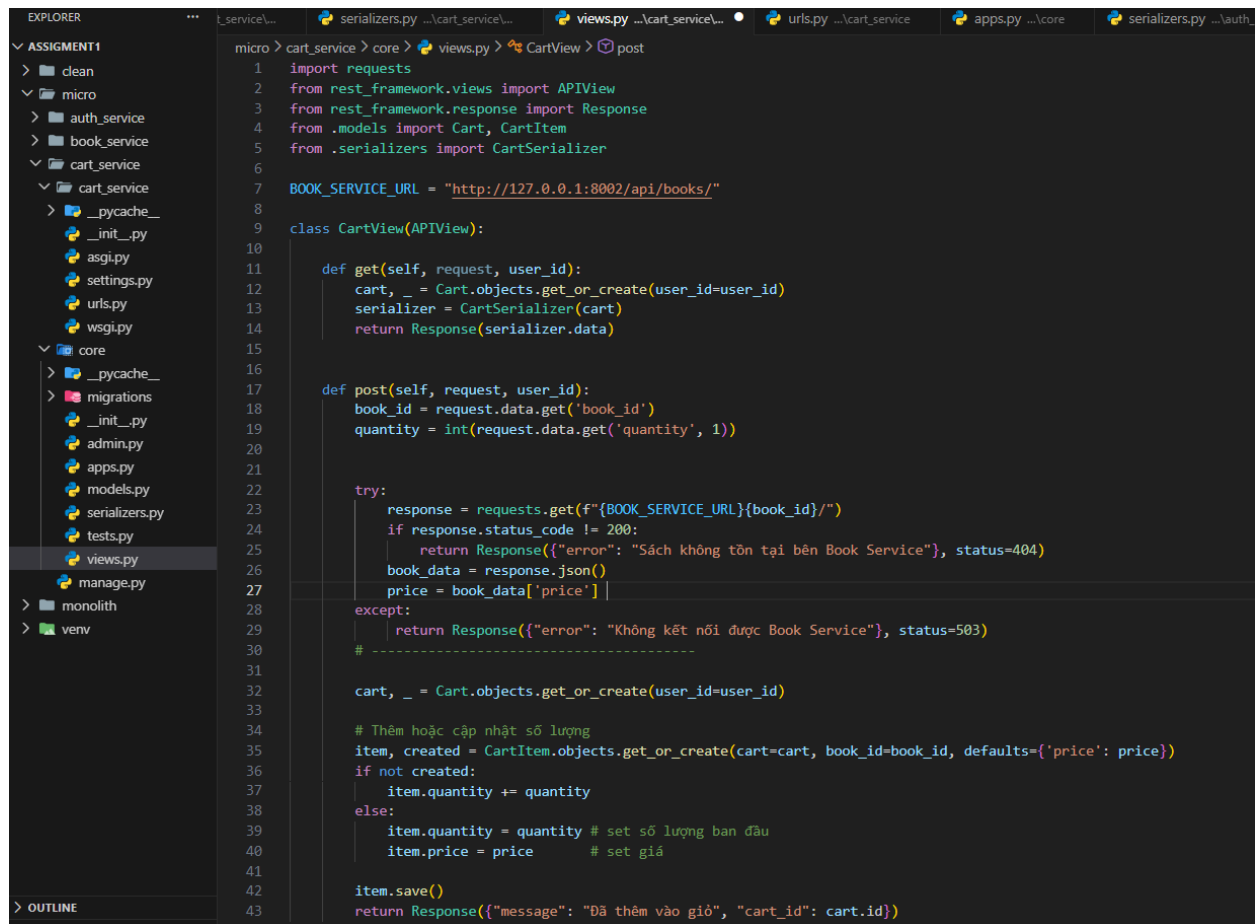
Hệ thống được phân rã thành 3 dịch vụ độc lập (Independent Services), mỗi dịch vụ chạy trên một tiến trình (process) và cổng (port) riêng biệt, giao tiếp với nhau thông qua RESTful APIs.

- 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.



```
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     def post(self, request, user_id):
17         book_id = request.data.get('book_id')
18         quantity = int(request.data.get('quantity', 1))
19
20         try:
21             response = requests.get(f"{BOOK_SERVICE_URL}{book_id}/")
22             if response.status_code != 200:
23                 return Response({"error": "Sách không tồn tại bên Book Service"}, status=404)
24             book_data = response.json()
25             price = book_data['price']
26
27         except:
28             return Response({"error": "Không kết nối được Book Service"}, status=503)
29         # -----
30
31         cart, _ = Cart.objects.get_or_create(user_id=user_id)
32
33         # Thêm hoặc cập nhật số lượng
34         item, created = CartItem.objects.get_or_create(cart=cart, book_id=book_id, defaults={'price': price})
35         if not created:
36             item.quantity += quantity
37         else:
38             item.quantity = quantity # set số lượng ban đầu
39             item.price = price # set giá
40
41         item.save()
42         return Response({"message": "Đã thêm vào giỏ", "cart_id": cart.id})
43
```

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)

The screenshot displays a REST client interface with the following details:

- Request Method:** POST
- Request URL:** http://127.0.0.1:8001/api/register/
- Request Body (JSON):**

```
{  "username": "micro_user",  "password": "123"}
```
- Response Status:** 201 Created
- Response Time:** 370 ms
- Response Size:** 355 B
- Response Body (JSON):**

```
{  "id": 2,  "username": "micro_user"}
```

## Xem Sách (Book Service - Port 8002)

The screenshot displays a REST client interface with a GET request to `http://127.0.0.1:8002/api/books/`. The request body is a JSON object representing a book:

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

The response status is `200 OK` with a response time of `13 ms` and a size of `407 B`. The response body is a JSON object representing a book:

```
1 {  
2   "id": 1,  
3   "title": "Doraemon",  
4   "author": "Fujiko",  
5   "price": "20000.00",  
6   "stock": 100  
7 }  
8  
9 }
```

## Tạo sách

The screenshot displays a REST client interface with a dark theme. At the top, the URL bar shows `http://127.0.0.1:8002/api/books/`. Below it, the method is set to **POST** and the same URL is entered in the request field. A **Send** button is on the right. The interface has tabs for **Params**, **Authorization**, **Headers (8)**, **Body** (selected), **Pre-request Script**, **Tests**, and **Settings**. Below these tabs, there are radio buttons for **none**, **form-data**, **x-www-form-urlencoded**, **raw** (selected), and **binary**, followed by a **JSON** dropdown and a **Beautify** button. The main area shows the request body as a JSON object: 

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

 Below this, there is a section for the response. It has tabs for **Body** (selected), **Cookies**, **Headers (10)**, and **Test Results**. To the right of these tabs, it shows a status of **201 Created**, a time of **12 ms**, and a size of **431 B**, along with a **Save Response** button. Below the tabs, there are buttons for **Pretty** (selected), **Raw**, **Preview**, and **Visualize**, followed by a **JSON** dropdown and a refresh icon. The response body is displayed as a JSON object: 

```
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 displays a REST client interface with a POST request to `http://127.0.0.1:8003/api/cart/1/`. The request body is a JSON object: `{"book_id": 1, "quantity": 2}`. The response is a 200 OK status with a response time of 38 ms and a body size of 376 B. The response body is a JSON object: `{"message": "Đã thêm vào giỏ", "cart_id": 1}`.

**Request:**

```
POST http://127.0.0.1:8003/api/cart/1/

{"book_id": 1, "quantity": 2}
```

**Response:**

```
200 OK 38 ms 376 B

{"message": "Đã thêm vào giỏ", "cart_id": 1}
```