

ASSIGNMENT 01

Hà Mạnh Dũng
MSV: B22DCCN125

Ngày 19 tháng 1 năm 2026

1 Tổng Quan

1.1 Giới Thiệu Dự Án

Dự án Book Store Web System được phát triển nhằm minh họa ba kiến trúc phần mềm khác nhau:

- **Monolithic Architecture:** Kiến trúc đơn thể, tất cả chức năng trong một ứng dụng
- **Clean Architecture:** Kiến trúc sạch với các lớp tách biệt rõ ràng
- **Microservices Architecture:** Kiến trúc vi dịch vụ với các service độc lập

1.2 Điểm Chung Giữa Các Version

1.2.1 Entities và Database Schema

Cả ba version đều sử dụng cùng một mô hình dữ liệu với 4 entities chính:

1. Customer: Quản lý thông tin khách hàng

- **id:** Khóa chính
- **name:** Tên khách hàng
- **email:** Email (unique)
- **password:** Mật khẩu đã được hash

2. Book: Quản lý thông tin sách

- **id:** Khóa chính
- **title:** Tiêu đề sách
- **author:** Tác giả
- **price:** Giá tiền
- **stock:** Số lượng tồn kho

3. Cart: Quản lý giỏ hàng

- `id`: Khóa chính
- `customer_id`: Tham chiếu đến Customer
- `created_at`: Thời gian tạo

4. **CartItem**: Quản lý các sản phẩm trong giỏ hàng

- `id`: Khóa chính
- `cart_id`: Tham chiếu đến Cart
- `book_id`: Tham chiếu đến Book
- `quantity`: Số lượng

1.2.2 Functional Requirements

Cả ba version đều implement các chức năng sau:

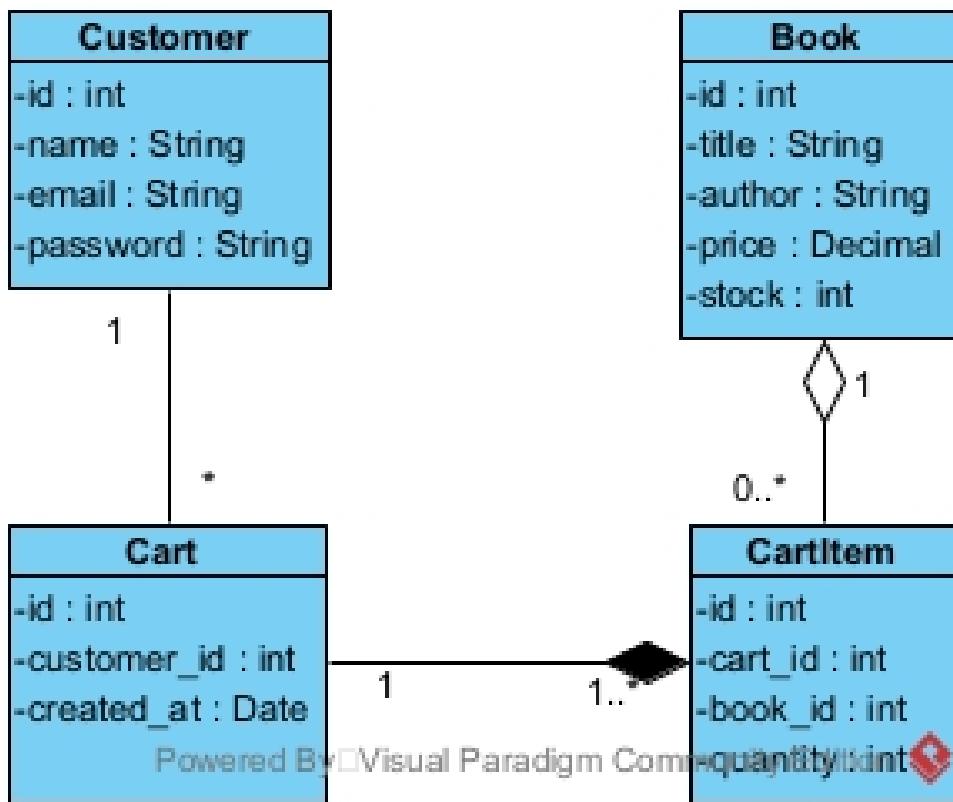
- **Customer Registration**: Đăng ký tài khoản khách hàng mới
- **Customer Login**: Đăng nhập với email và password
- **View Book Catalog**: Xem danh sách tất cả sách
- **Add to Cart**: Thêm sách vào giỏ hàng
- **View Cart**: Xem nội dung giỏ hàng

1.2.3 Technology Stack

- **Backend Framework**: Django 4.2.7
- **Database**: MySQL
- **API Framework**: Django REST Framework
- **Authentication**: Django password hashing
- **CORS**: django-cors-headers

1.2.4 Class Diagram

Cả ba version đều sử dụng cùng một Class Diagram để mô tả mối quan hệ giữa các entities:



Hình 1: Class Diagram - Mối quan hệ giữa các entities

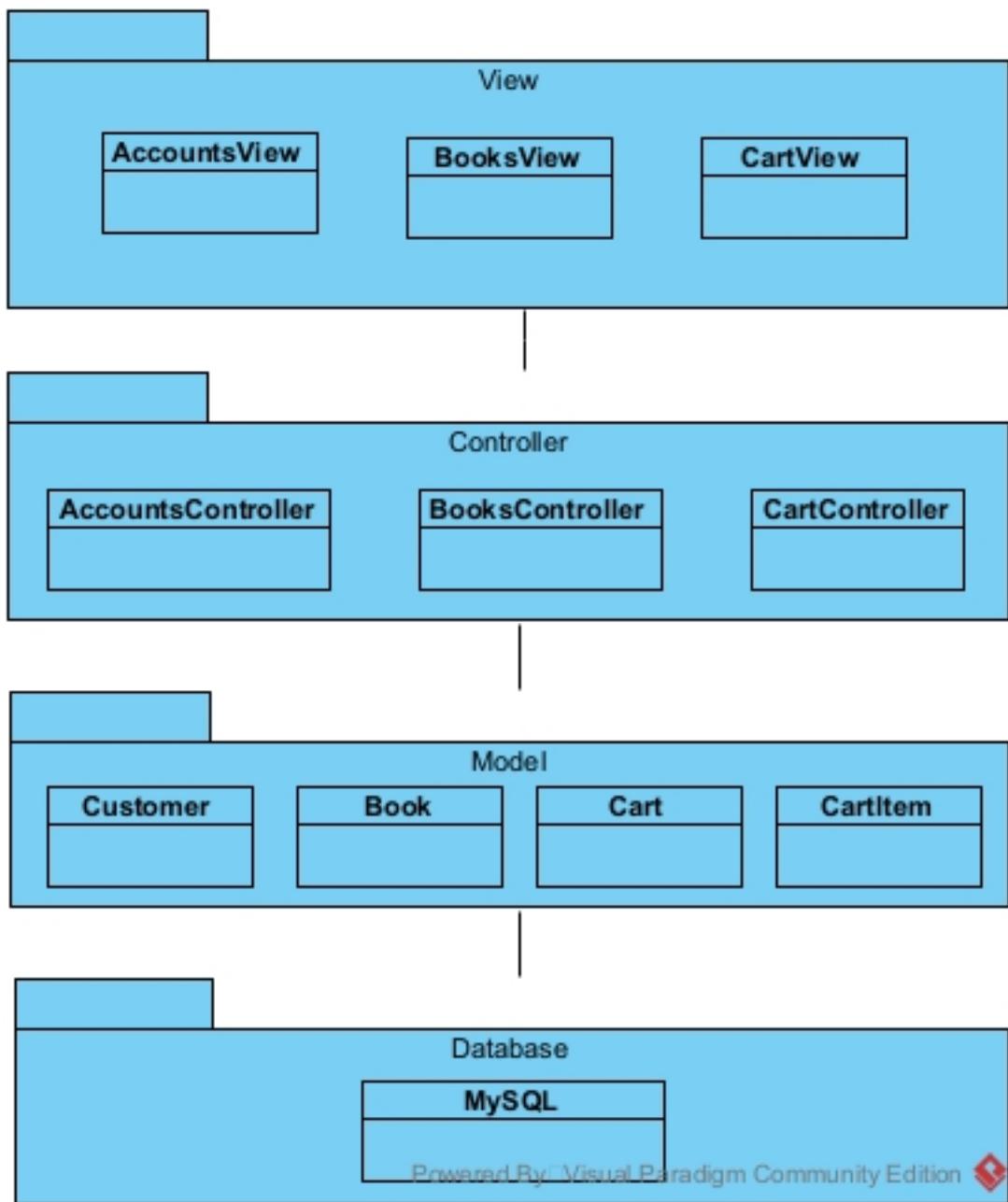
Giải thích các mối quan hệ:

- **Customer → Cart:** Association (1 to 0..*) - Một khách hàng có thể có nhiều giỏ hàng
- **Cart → CartItem:** Composition (1 to 1..*) - CartItem không thể tồn tại độc lập, phụ thuộc vào Cart
- **Book → CartItem:** Aggregation (1 to 0..*) - Book có thể tồn tại độc lập, CartItem chỉ tham chiếu đến Book

2 Version A - Monolithic Architecture

2.1 Tổng Quan Kiến Trúc

Kiến trúc Monolithic là kiến trúc truyền thống, tất cả các chức năng được đặt trong một ứng dụng Django duy nhất. Hệ thống được tổ chức theo mô hình MVC (Model-View-Controller).



Hình 2: MVC Layer Diagram - Monolithic Architecture

2.2 Cấu Trúc Dự Án

Dự án được tổ chức thành các Django apps:

- accounts/: Quản lý khách hàng (registration, login)

- books/: Quản lý danh mục sách
- cart/: Quản lý giỏ hàng
- bookstore/: Cấu hình chính của Django project

2.3 Giải Thích Code

2.3.1 Models Layer

Customer Model (accounts/models.py):

```

1 //Ha Manh Dung - B22DCCN125
2 class Customer(models.Model):
3     id = models.AutoField(primary_key=True)
4     name = models.CharField(max_length=255)
5     email = models.EmailField(unique=True)
6     password = models.CharField(max_length=255)
7
8     def set_password(self, raw_password):
9         self.password = make_password(raw_password)
10
11    def check_password(self, raw_password):
12        return check_password(raw_password, self.password)

```

Giải thích:

- Model Customer định nghĩa cấu trúc bảng customers trong database
- `set_password()`: Hash password trước khi lưu vào database
- `check_password()`: Xác thực password khi đăng nhập

Cart và CartItem Models (cart/models.py):

```

1 //Ha Manh Dung - B22DCCN125
2 class Cart(models.Model):
3     id = models.AutoField(primary_key=True)
4     customer = models.ForeignKey(Customer,
5                                     on_delete=models.CASCADE,
6                                     related_name='carts')
7     created_at = models.DateTimeField(auto_now_add=True)
8
9 class CartItem(models.Model):
10    id = models.AutoField(primary_key=True)
11    cart = models.ForeignKey(Cart,
12                             on_delete=models.CASCADE,
13                             related_name='items')
14    book = models.ForeignKey(Book,
15                             on_delete=models.CASCADE)
16    quantity = models.IntegerField(default=1)

```

Giải thích:

- Cart có ForeignKey đến Customer với CASCADE delete

- CartItem có ForeignKey đến cả Cart và Book
- Khi Cart bị xóa, tất cả CartItem liên quan cũng bị xóa (CASCADE)

2.3.2 Views Layer (Controller)

Customer Registration (accounts/views.py):

```

1 //Ha Manh Dung - B22DCCN125
2 @api_view(['POST'])
3 def register(request):
4     serializer = CustomerSerializer(data=request.data)
5     if serializer.is_valid():
6         customer = serializer.save()
7         response_serializer = CustomerResponseSerializer(customer)
8         return Response(response_serializer.data,
9                         status=status.HTTP_201_CREATED)
10    return Response(serializer.errors,
11                    status=status.HTTP_400_BAD_REQUEST)

```

Giải thích:

- Sử dụng decorator @api_view để xử lý HTTP POST request
- CustomerSerializer validate và tạo Customer mới
- CustomerResponseSerializer trả về dữ liệu không bao gồm password
- Trả về HTTP 201 nếu thành công, 400 nếu có lỗi validation

Add to Cart (cart/views.py):

```

1 //Ha Manh Dung - B22DCCN125
2 @api_view(['POST'])
3 def add_to_cart(request, customer_id):
4     # Validate customer exists
5     try:
6         customer = Customer.objects.get(id=customer_id)
7     except Customer.DoesNotExist:
8         return Response({'error': 'Customer not found'},
9                         status=status.HTTP_404_NOT_FOUND)
10
11    # Validate book exists and stock
12    book = Book.objects.get(id=book_id)
13    if book.stock < quantity:
14        return Response({'error': 'Insufficient stock'},
15                        status=status.HTTP_400_BAD_REQUEST)
16
17    # Get or create cart
18    cart, created = Cart.objects.get_or_create(customer=customer)
19
20    # Get or create cart item
21    cart_item, item_created = CartItem.objects.get_or_create(
22        cart=cart, book=book, defaults={'quantity': quantity})

```

```

23     )
24
25     if not item_created:
26         cart_item.quantity += quantity
27         cart_item.save()

```

Giải thích:

- Validate customer và book tồn tại trước khi thêm vào cart
- Kiểm tra số lượng tồn kho đủ không
- Sử dụng `get_or_create()` để tạo cart nếu chưa có
- Nếu CartItem đã tồn tại, tăng quantity thay vì tạo mới

2.3.3 Serializers Layer

CustomerSerializer (`accounts/serializers.py`):

```

1 //Ha Manh Dung - B22DCCN125
2 class CustomerSerializer(serializers.ModelSerializer):
3     password = serializers.CharField(write_only=True,
4                                     required=True)
5
6     class Meta:
7         model = Customer
8         fields = ['id', 'name', 'email', 'password']
9         extra_kwargs = {
10             'password': {'write_only': True}
11         }
12
13     def create(self, validated_data):
14         customer = Customer.objects.create(
15             name=validated_data['name'],
16             email=validated_data['email']
17         )
18         customer.set_password(validated_data['password'])
19         customer.save()
20         return customer

```

Giải thích:

- `write_only=True`: Password chỉ nhận vào, không trả về trong response
- Override method `create()` để hash password trước khi lưu
- Sử dụng `set_password()` để đảm bảo password được hash đúng cách

2.3.4 URL Routing

Main URLs (`bookstore/urls.py`):

```

1 //Ha Manh Dung - B22DCCN125
2 urlpatterns = [
3     path('admin/', admin.site.urls),
4     path('api/accounts/', include('accounts.urls')),
5     path('api/books/', include('books.urls')),
6     path('api/cart/', include('cart.urls')),
7 ]

```

Giải thích:

- Tổ chức URLs theo từng app
- Mỗi app có file `urls.py` riêng để quản lý routing
- Sử dụng `include()` để include các URL patterns từ apps

2.4 Ưu và Nhược Điểm

Ưu điểm:

- Đơn giản, dễ phát triển ban đầu
- Dễ test và debug
- Không cần xử lý inter-service communication
- Phù hợp với ứng dụng nhỏ và vừa

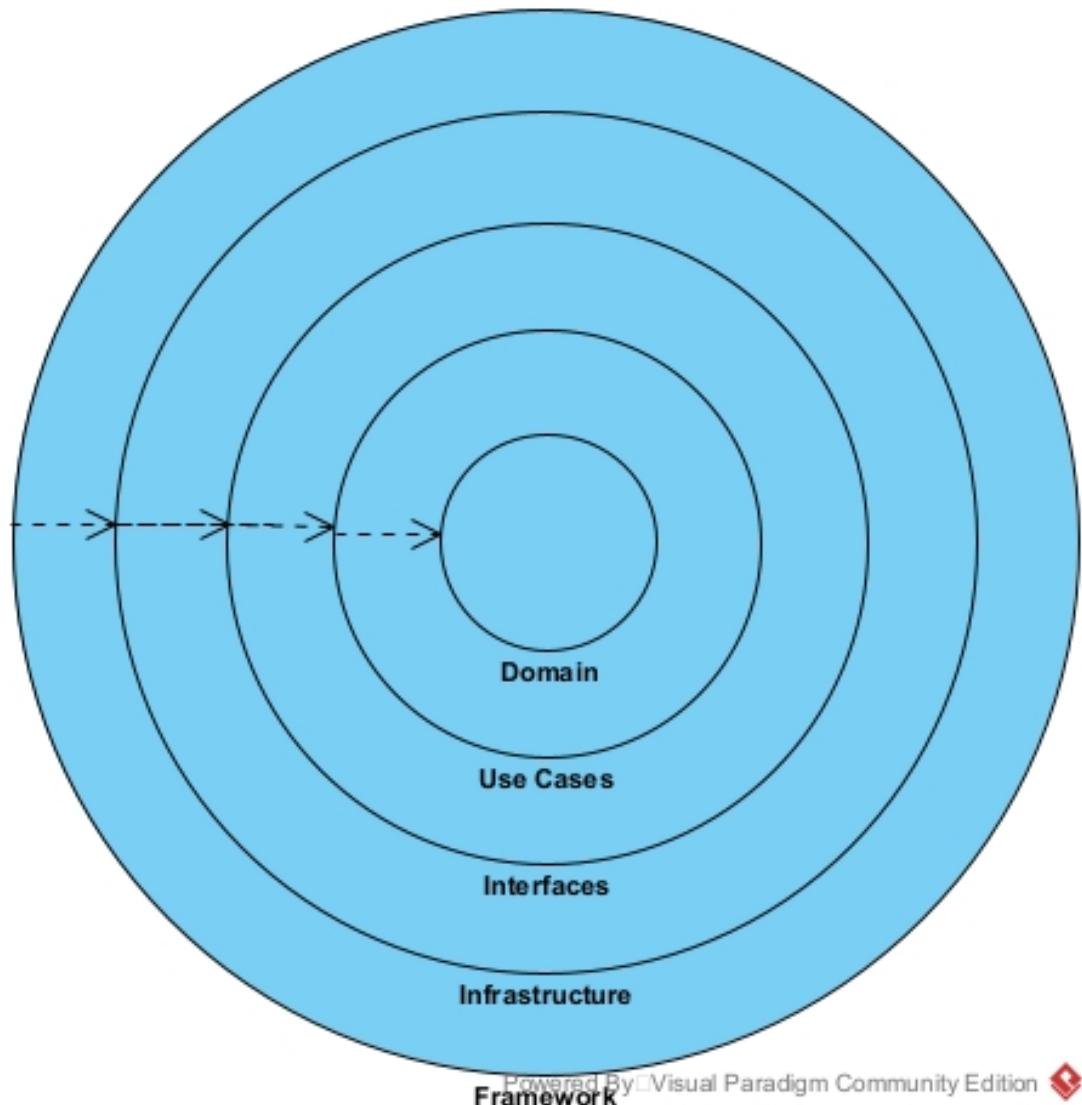
Nhược điểm:

- Khó scale khi ứng dụng lớn
- Tất cả code trong một codebase, khó maintain
- Một lỗi có thể ảnh hưởng toàn bộ hệ thống
- Khó deploy từng phần riêng biệt

3 Version B - Clean Architecture

3.1 Tổng Quan Kiến Trúc

Clean Architecture tách biệt business logic khỏi framework và infrastructure. Kiến trúc này tuân theo Dependency Inversion Principle, các dependencies chỉ hướng vào trong.



Hình 3: Clean Architecture Diagram

3.2 Cấu Trúc Dự Án

Dự án được tổ chức thành 5 layers:

- `domain/`: Entities và Repository interfaces (trung tâm)
- `usecases/`: Business logic use cases
- `interfaces/`: Serializers và adapters
- `infrastructure/`: Repository implementations

- `framework/`: Django framework layer (ngoài cùng)

3.3 Giải Thích Code

3.3.1 Domain Layer

Entities (`domain/entities.py`):

```

1 //Ha Manh Dung - B22DCCN125
2 @dataclass
3 class Customer:
4     """Customer entity"""
5     id: Optional[int]
6     name: str
7     email: str
8     password: str
9
10    def __post_init__(self):
11        if self.id is None:
12            self.id = 0

```

Giải thích:

- Sử dụng `@dataclass` để định nghĩa entities đơn giản
- Entities không phụ thuộc vào framework hay database
- Chứa pure business objects, không có logic phức tạp

Repository Interfaces (`domain/repositories.py`):

```

1 //Ha Manh Dung - B22DCCN125
2 class ICustomerRepository(ABC):
3     """Customer repository interface"""
4
5     @abstractmethod
6     def create(self, customer: Customer) -> Customer:
7         pass
8
9     @abstractmethod
10    def get_by_id(self, customer_id: int) -> Optional[Customer]:
11        pass
12
13    @abstractmethod
14    def get_by_email(self, email: str) -> Optional[Customer]:
15        pass

```

Giải thích:

- Định nghĩa contracts (interfaces) cho data access
- Sử dụng ABC (Abstract Base Class) để đảm bảo implementation
- Domain layer không biết cách implement, chỉ biết interface

3.3.2 Use Cases Layer

RegisterCustomerUseCase (usecases/customer_usecases.py):

```
1 //Ha Manh Dung - B22DCCN125
2 class RegisterCustomerUseCase:
3     """Register a new customer"""
4
5     def __init__(self, customer_repository: ICustomerRepository):
6         self.customer_repository = customer_repository
7
8     def execute(self, name: str, email: str, password: str) ->
9         Customer:
10        # Check if email already exists
11        existing = self.customer_repository.get_by_email(email)
12        if existing:
13            raise ValueError("Email already exists")
14
15        # Create customer with hashed password
16        hashed_password = make_password(password)
17        customer = Customer(
18            id=None, name=name, email=email,
19            password=hashed_password
20        )
21
22        return self.customer_repository.create(customer)
```

Giải thích:

- Use case nhận repository interface, không phụ thuộc vào implementation
- Chứa business logic: kiểm tra email trùng, hash password
- Có thể test độc lập mà không cần Django
- Method `execute()` thực hiện một use case cụ thể

AddToCartUseCase (usecases/cart_usecases.py):

```
1 //Ha Manh Dung - B22DCCN125
2 class AddToCartUseCase:
3     def __init__(self, cart_repo, cart_item_repo,
4                  customer_repo, book_repo):
5         self.cart_repository = cart_repo
6         self.cart_item_repository = cart_item_repo
7         self.customer_repository = customer_repo
8         self.book_repository = book_repo
9
10    def execute(self, customer_id: int, book_id: int,
11                quantity: int) -> Cart:
12        # Validate customer exists
13        customer = self.customer_repository.get_by_id(customer_id)
14        if not customer:
15            raise ValueError("Customer not found")
```

```

16
17     # Validate book exists and has stock
18     book = self.book_repository.get_by_id(book_id)
19     if not book:
20         raise ValueError("Book not found")
21     if book.stock < quantity:
22         raise ValueError("Insufficient stock")
23
24     # Get or create cart
25     cart = self.cart_repository.get_or_create_by_customer_id(
26         customer_id
27     )
28
29     # Get or create cart item
30     cart_item = self.cart_item_repository.get_or_create(
31         cart_id=cart.id, book_id=book_id, quantity=quantity
32     )
33
34     return cart

```

Giải thích:

- Use case phức tạp hơn, cần nhiều repositories
- Tập trung business logic: validation, business rules
- Throw exceptions thay vì trả về error codes
- Dễ test vì không phụ thuộc vào framework

3.3.3 Infrastructure Layer

`CustomerRepository` (`infrastructure/repositories.py`):

```

1 //Ha Manh Dung - B22DCCN125
2 class CustomerRepository(ICustomerRepository):
3     """Customer repository implementation"""
4
5     def create(self, customer: Customer) -> Customer:
6         model = CustomerModel.objects.create(
7             name=customer.name,
8             email=customer.email,
9             password=customer.password
10        )
11        return Customer(
12            id=model.id,
13            name=model.name,
14            email=model.email,
15            password=model.password
16        )
17
18    def get_by_email(self, email: str) -> Optional[Customer]:
19        try:

```

```

20         model = CustomerModel.objects.get(email=email)
21     return Customer(
22         id=model.id,
23         name=model.name,
24         email=model.email,
25         password=model.password
26     )
27 except CustomerModel.DoesNotExist:
28     return None

```

Giải thích:

- Implement interface từ domain layer
- Sử dụng Django ORM (CustomerModel) để truy cập database
- Chuyển đổi giữa Domain entities và Django models
- Xử lý exceptions và trả về None nếu không tìm thấy

3.3.4 Framework Layer

Views (framework/views.py):

```

1 //Ha Manh Dung - B22DCCN125
2 # Initialize repositories
3 customer_repo = CustomerRepository()
4 book_repo = BookRepository()
5 cart_repo = CartRepository()
6 cart_item_repo = CartItemRepository()

7
8 @api_view(['POST'])
9 def register(request):
10     try:
11         name = request.data.get('name')
12         email = request.data.get('email')
13         password = request.data.get('password')

14         if not all([name, email, password]):
15             return Response(
16                 {'error': 'Missing required fields'},
17                 status=status.HTTP_400_BAD_REQUEST
18             )
19
20         use_case = RegisterCustomerUseCase(customer_repo)
21         customer = use_case.execute(name, email, password)

22         return Response(
23             CustomerSerializer.to_dict(customer),
24             status=status.HTTP_201_CREATED
25         )
26     except ValueError as e:
27         return Response(

```

```
30         { 'error': str(e)},
31         status=status.HTTP_400_BAD_REQUEST
32     )
```

Giải thích:

- Views là entry point, xử lý HTTP requests
- Khởi tạo repositories và use cases
- Gọi use case để thực hiện business logic
- Chuyển đổi entities thành JSON response qua serializers
- Xử lý exceptions và trả về HTTP status codes phù hợp

3.4 Dependency Flow

Dependencies chỉ hướng vào trong:

1. **Framework** → Infrastructure, Interfaces, Use Cases, Domain
2. **Infrastructure** → Domain (implements interfaces)
3. **Interfaces** → Use Cases, Domain
4. **Use Cases** → Domain
5. **Domain** → Không phụ thuộc gì

3.5 Ưu và Nhược Điểm

Ưu điểm:

- Business logic độc lập với framework
- Dễ test, có thể test use cases không cần Django
- Dễ maintain, thay đổi framework không ảnh hưởng business logic
- Tuân theo SOLID principles

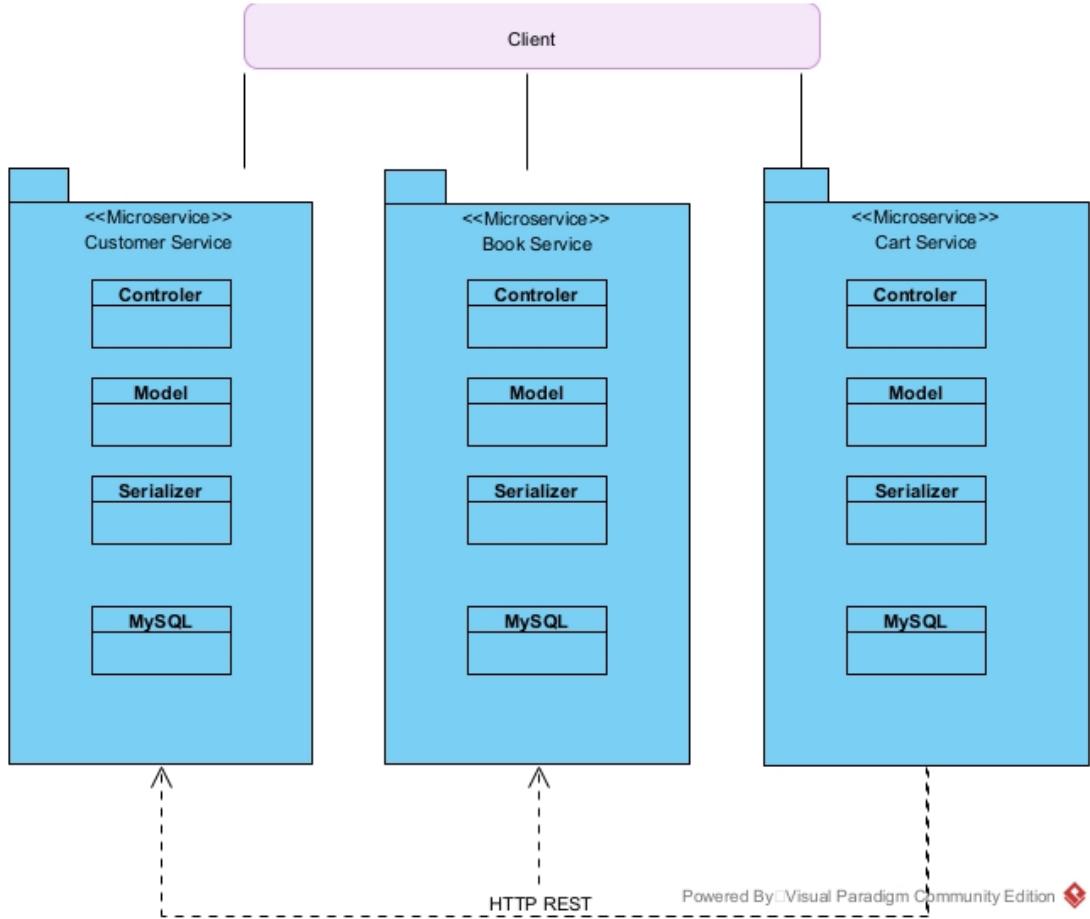
Nhược điểm:

- Phức tạp hơn Monolithic
- Nhiều layers, nhiều files
- Có thể over-engineering cho dự án nhỏ
- Cần hiểu rõ Clean Architecture principles

4 Version C - Microservices Architecture

4.1 Tổng Quan Kiến Trúc

Microservices Architecture chia hệ thống thành các services độc lập, mỗi service có database riêng và giao tiếp qua REST APIs.



Hình 4: Microservices Architecture Diagram

4.2 Các Services

Hệ thống được chia thành 3 services:

1. **Customer Service** (Port 8001): Quản lý khách hàng
2. **Book Service** (Port 8002): Quản lý danh mục sách
3. **Cart Service** (Port 8003): Quản lý giỏ hàng

4.3 Giải Thích Code

4.3.1 Customer Service

Model (`customers/models.py`):

```

1 //Ha Manh Dung - B22DCCN125
2 class Customer(models.Model):
3     id = models.AutoField(primary_key=True)
4     name = models.CharField(max_length=255)
5     email = models.EmailField(unique=True)
6     password = models.CharField(max_length=255)
7
8     def set_password(self, raw_password):
9         self.password = make_password(raw_password)
10
11    def check_password(self, raw_password):
12        return check_password(raw_password, self.password)

```

View (customers/views.py):

```

1 //Ha Manh Dung - B22DCCN125
2 @api_view(['GET'])
3 def get_customer(request, customer_id):
4     """Get customer by ID (for other services)"""
5     try:
6         customer = Customer.objects.get(id=customer_id)
7         serializer = CustomerResponseSerializer(customer)
8         return Response(serializer.data,
9                         status=status.HTTP_200_OK)
10    except Customer.DoesNotExist:
11        return Response({'error': 'Customer not found'},
12                        status=status.HTTP_404_NOT_FOUND)

```

Giải thích:

- Customer Service có endpoint GET /api/customers/<id>/ để các service khác gọi
- Trả về thông tin customer dạng JSON
- Sử dụng cho inter-service communication

4.3.2 Book Service

View (books/views.py):

```

1 //Ha Manh Dung - B22DCCN125
2 @api_view(['GET'])
3 def get_books_by_ids(request):
4     """Get multiple books by IDs (for other services)"""
5     book_ids = request.GET.getlist('ids')
6     if not book_ids:
7         return Response({'error': 'No book IDs provided'},
8                         status=status.HTTP_400_BAD_REQUEST)
9
10    try:
11        book_ids = [int(id) for id in book_ids]
12        books = Book.objects.filter(id__in=book_ids)
13        serializer = BookSerializer(books, many=True)

```

```

14     return Response(serializer.data,
15         status=status.HTTP_200_OK)
16 except ValueError:
17     return Response({'error': 'Invalid book IDs'},
18                     status=status.HTTP_400_BAD_REQUEST)

```

Giải thích:

- Endpoint GET /api/books/batch/?ids=1&ids=2 để lấy nhiều books cùng lúc
- Tối ưu cho trường hợp Cart Service cần lấy thông tin nhiều books
- Giảm số lượng HTTP requests

4.3.3 Cart Service

Service Clients (carts/services.py):

```

1 //Ha Manh Dung - B22DCCN125
2 class CustomerService:
3     """Client for customer-service"""
4
5     @staticmethod
6     def get_customer(customer_id):
7         """Get customer from customer-service"""
8         try:
9             url =
10                f'{settings.CUSTOMER_SERVICE_URL}/api/customers/{customer_id}'
11            response = requests.get(url, timeout=5)
12            if response.status_code == 200:
13                return response.json()
14            return None
15        except requests.RequestException:
16            return None
17
18 class BookService:
19     """Client for book-service"""
20
21     @staticmethod
22     def get_book(book_id):
23         """Get book from book-service"""
24         try:
25             url =
26                f'{settings.BOOK_SERVICE_URL}/api/books/{book_id}/'
27            response = requests.get(url, timeout=5)
28            if response.status_code == 200:
29                return response.json()
30            return None
31        except requests.RequestException:
32            return None
33
34 @staticmethod
35     def get_books_by_ids(book_ids):
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
617
618
619
619
620
621
622
623
624
625
626
627
627
628
629
629
630
631
632
633
634
635
635
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
15
```

```

34     """Get multiple books from book-service"""
35     try:
36         url = f"{settings.BOOK_SERVICE_URL}/api/books/batch/"
37         params = [({'ids': book_id}) for book_id in book_ids]
38         response = requests.get(url, params=params, timeout=5)
39         if response.status_code == 200:
40             return response.json()
41         return []
42     except requests.RequestException:
43         return []

```

Giải thích:

- Service clients là HTTP clients để giao tiếp với các services khác
- Sử dụng thư viện `requests` để gọi REST APIs
- Có timeout để tránh blocking quá lâu
- Xử lý exceptions và trả về `None/[]` nếu có lỗi

Add to Cart View (`carts/views.py`):

```

1 // Ha Manh Dung - B22DCCN125
2 @api_view(['POST'])
3 def add_to_cart(request, customer_id):
4     """Add books to the shopping cart"""
5     # Validate customer exists via customer-service
6     customer_data = CustomerService.get_customer(customer_id)
7     if not customer_data:
8         return Response({'error': 'Customer not found'},
9                         status=status.HTTP_404_NOT_FOUND)
10
11    # Validate book exists and has stock via book-service
12    book_data = BookService.get_book(book_id)
13    if not book_data:
14        return Response({'error': 'Book not found'},
15                        status=status.HTTP_404_NOT_FOUND)
16
17    if book_data['stock'] < quantity:
18        return Response({'error': 'Insufficient stock'},
19                        status=status.HTTP_400_BAD_REQUEST)
20
21    # Get or create cart for customer
22    cart, created = Cart.objects.get_or_create(
23        customer_id=customer_id
24    )
25
26    # Get or create cart item
27    cart_item, item_created = CartItem.objects.get_or_create(
28        cart=cart, book_id=book_id, defaults={'quantity':
29            quantity}

```

```

30
31     # Fetch cart items with book data
32     items = CartItem.objects.filter(cart=cart)
33     book_ids = [item.book_id for item in items]
34     books_data = BookService.get_books_by_ids(book_ids)
35     books_dict = {book['id']: book for book in books_data}
36
37     # Attach book data to items
38     for item in items:
39         item._book_data = books_dict.get(item.book_id)
40
41     cart._customer_data = customer_data
42
43     serializer = CartSerializer(cart)
44     return Response(serializer.data, status=status.HTTP_200_OK)

```

Giải thích:

- Cart Service gọi Customer Service để validate customer
- Gọi Book Service để validate book và kiểm tra stock
- Lưu cart và cart items vào database riêng của Cart Service
- Fetch thông tin books từ Book Service để trả về đầy đủ
- Attach data vào objects để serializer có thể serialize

4.3.4 Database per Service

Mỗi service có database riêng:

- bookstore_customer: Database cho Customer Service
- bookstore_book: Database cho Book Service
- bookstore_cart: Database cho Cart Service

Giải thích:

- Mỗi service độc lập về dữ liệu
- Có thể scale từng service riêng biệt
- Không có shared database, tránh tight coupling
- Mỗi service có thể chọn database phù hợp

4.4 Inter-Service Communication

- **Cart Service → Customer Service:**
 - Purpose: Validate customer exists
 - Method: HTTP GET
 - Endpoint: `http://localhost:8001/api/customers/<id>/`
- **Cart Service → Book Service:**
 - Purpose: Get book information and validate stock
 - Method: HTTP GET
 - Endpoints:
 - * `http://localhost:8002/api/books/<id>/`
 - * `http://localhost:8002/api/books/batch/?ids=1&ids=2`

4.5 Ưu và Nhược Điểm

Ưu điểm:

- Mỗi service có thể phát triển và deploy độc lập
- Có thể scale từng service riêng biệt
- Fault isolation: lỗi một service không ảnh hưởng service khác
- Có thể sử dụng công nghệ khác nhau cho từng service
- Phù hợp với team lớn, mỗi team phụ trách một service

Nhược điểm:

- Phức tạp hơn về infrastructure và deployment
- Cần xử lý distributed transactions
- Network latency giữa các services
- Khó debug khi có lỗi liên quan đến nhiều services
- Cần service discovery và load balancing

5 So Sánh Ba Kiến Trúc

5.1 Bảng So Sánh

Tiêu chí	Monolithic	Clean Architecture	Microservices
Độ phức tạp	Thấp	Trung bình	Cao
Dễ phát triển	Rất dễ	Dễ	Khó
Dễ test	Dễ	Rất dễ	Khó
Khả năng scale	Thấp	Trung bình	Cao
Maintainability	Trung bình	Cao	Trung bình
Deployment	Dơn giản	Dơn giản	Phức tạp
Fault isolation	Thấp	Trung bình	Cao
Technology diversity	Không	Không	Có

Bảng 1: So sánh ba kiến trúc

5.2 Khi Nào Sử Dụng

Monolithic Architecture:

- Ứng dụng nhỏ và vừa
- Team nhỏ
- Cần phát triển nhanh
- Không cần scale cao

Clean Architecture:

- Ứng dụng cần maintain lâu dài
- Cần business logic phức tạp
- Có thể thay đổi framework trong tương lai
- Cần test coverage cao

Microservices Architecture:

- Ứng dụng lớn, nhiều teams
- Cần scale cao
- Các services có thể scale độc lập
- Cần fault isolation
- Có thể sử dụng công nghệ khác nhau

6 Kết Luận

Dự án Book Store Web System đã thành công implement ba kiến trúc phần mềm khác nhau:

1. **Monolithic Architecture**: Đơn giản, phù hợp cho ứng dụng nhỏ
2. **Clean Architecture**: Tách biệt business logic, dễ maintain và test
3. **Microservices Architecture**: Phù hợp cho hệ thống lớn, có thể scale độc lập

Mỗi kiến trúc có ưu nhược điểm riêng và phù hợp với các tình huống khác nhau. Việc hiểu rõ từng kiến trúc giúp lựa chọn giải pháp phù hợp cho từng dự án cụ thể.

7 Tài Liệu Tham Khảo

- Django Documentation: <https://docs.djangoproject.com/>
- Django REST Framework: <https://www.django-rest-framework.org/>
- Clean Architecture - Robert C. Martin
- Microservices Patterns - Chris Richardson
- MySQL Documentation: <https://dev.mysql.com/doc/>