Dokumentasi API Sebelum JWT Authentication

Gambaran Umum

API Item Management ini dibangun menggunakan FastAPI dengan pendekatan Domain-Driven Design (DDD). Sebelum implementasi JWT, API ini tidak memiliki sistem autentikasi dan otorisasi.

Instalasi

Prasyarat

• Python 3.8 atau lebih baru

• pip (Python package manager)

• Git (optional, untuk clone repository)

Langkah 1: Setup Project

Clone Repository (Jika dari GitHub):

```
git clone https://github.com/DoItMark/TST-API-DDD.git
cd TST-API-DDD
```

Atau Buat Directory Baru:

```
mkdir item-management-api
cd item-management-api
```

Langkah 2: Install Dependencies

File requirements.txt berisi:

```
fastapi==0.104.1
uvicorn[standard]==0.24.0
pydantic==2.5.0
```

Install packages:

```
pip install -r requirements.txt
```

Langkah 3: Jalankan API Server

Menggunakan Python:

```
python listing_api.py
```

Atau menggunakan Uvicorn:

```
uvicorn listing_api:app --reload
```

Akses API Documentation:

• Swagger UI: http://localhost:8000/docs

• ReDoc: http://localhost:8000/redoc

Struktur Kode

1. Import dan Dependencies

```
from fastapi import FastAPI, HTTPException, status
from pydantic import BaseModel, Field
from typing import List, Optional
from enum import Enum
from uuid import UUID, uuid4
from decimal import Decimal
from datetime import datetime
```

Penjelasan:

• `FastAPI`: Framework web untuk membangun API

• `Pydantic`: Library untuk validasi data menggunakan Python type hints

• `UUID`: Untuk generate unique identifier

• `Decimal`: Untuk perhitungan harga yang akurat

2. Enums dan Value Objects

```
class ItemState(str, Enum):
    ACTIVE = "Active"
    SOLD = "Sold"
    DELISTED = "Delisted"

class Money(BaseModel):
    amount: Decimal
    currency: str = "USD"
```

```
class Condition(BaseModel):
    score: int = Field(..., ge=1, le=10)
    detailed_description: str
    known_defects: List[str] = Field(default_factory=list)
```

Penjelasan:

• `ItemState`: Enum untuk status listing (Aktif, Terjual, Dihapus)

• `Money`: Value Object untuk representasi harga yang immutable

• `Condition`: Value Object untuk kondisi barang dengan skor 1-10

3. Aggregate Root - Listing

```
class Listing(BaseModel):
    listing_id: UUID = Field(default_factory=uuid4)
    seller_id: UUID
    title: str
    item_state: ItemState = Field(default=ItemState.ACTIVE)
    price: Money
    condition: Condition
    attributes: List[Attribute] = Field(default_factory=list)
    created_at: datetime = Field(default_factory=datetime.utcnow)
    updated_at: datetime = Field(default_factory=datetime.utcnow)

    def activate_listing(self) -> None:
        """Aktivasi listing"""
        if self.item_state == ItemState.SOLD:
            raise ValueError("Cannot activate a sold listing")
        self.item_state = ItemState.ACTIVE
        self.updated_at = datetime.utcnow()

    def delist_listing(self, reason: DelistReason) -> None:
        """Hapus listing dengan alasan"""
        if self.item_state == ItemState.DELISTED:
            raise ValueError("Listing is already delisted")
        self.item_state = ItemState.DELISTED
        self.updated_at = datetime.utcnow()

    def update_price(self, new_price: Money) -> None:
        """Update harga listing"""
        if self.item_state == ItemState.SOLD:
            raise ValueError("Cannot update price of a sold listing")
        self.price = new_price
        self.updated_at = datetime.utcnow()
```

Penjelasan:

• `Listing` adalah Aggregate Root yang menjadi source of truth

• Memiliki business logic untuk activate, delist, dan update price

• Mencegah operasi ilegal (misalnya aktivasi listing yang sudah terjual)

4. Storage (In-Memory)

```
listings_db: dict[UUID, Listing] = {}
search_index_db: dict[UUID, SearchIndexModel] = {}
```

Penjelasan:

• Menggunakan dictionary Python untuk penyimpanan sementara

• `listings_db`: Menyimpan semua listing

• `search_index_db`: Menyimpan index untuk pencarian

5. Endpoints API

Create Listing (TANPA AUTENTIKASI)

```
@app.post("/listings", response_model=ListingResponse, status_code=status.HTTP_201_CR
  EATED)
async def create_listing(request: CreateListingRequest):
    """Create a new listing"""
    listing = Listing(
        seller_id=request.seller_id,  # Seller ID dari request
        title=request.title,
        price=request.price,
        condition=request.condition,
        attributes=request.attributes or []
    )

    listings_db[listing.listing_id] = listing
    return listing
```

Masalah:

• ✗ Tidak ada autentikasi - siapa saja bisa membuat listing

• ✗ Tidak ada verifikasi identitas seller

• ✗ User bisa mengaku sebagai seller lain dengan mengirim seller_id berbeda

Update Price (TANPA AUTENTIKASI)

```
@app.patch("/listings/{listing_id}/price", response_model=ListingResponse)
async def update_listing_price(listing_id: UUID, request: UpdatePriceRequest):
    """Update listing price"""
    if listing_id not in listings_db:
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND,
            detail=f"Listing {listing_id} not found"
        )

    listing = listings_db[listing_id]
    listing.update_price(request.new_price)
    return listing
```

Masalah:

• ✗ Siapa saja bisa mengubah harga listing orang lain

• ✗ Tidak ada ownership check

• ✗ Tidak ada authorization

Delete Listing (TANPA AUTENTIKASI)

```
@app.delete("/listings/{listing_id}", status_code=status.HTTP_204_NO_CONTENT)
async def delete_listing(listing_id: UUID):
    """Delete a listing"""
    if listing_id not in listings_db:
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND,
            detail=f"Listing {listing_id} not found"
        )

    del listings_db[listing_id]
```

Masalah:

• ✗ Siapa saja bisa menghapus listing orang lain

• ✗ Tidak ada verifikasi kepemilikan

6. Read Endpoints (Public)

```
@app.get("/listings/{listing_id}", response_model=ListingResponse)
async def get_listing(listing_id: UUID):
    """Get a listing by ID"""
    if listing_id not in listings_db:
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND,
            detail=f"Listing {listing_id} not found"
        )
    return listings_db[listing_id]

@app.get("/listings", response_model=List[ListingResponse])
async def list_listings(
    seller_id: Optional[UUID] = None,
    item_state: Optional[ItemState] = None,
    skip: int = 0,
    limit: int = 100
):
    """List all listings with optional filters"""
    filtered_listings = list(listings_db.values())

    if seller_id:
        filtered_listings = [l for l in filtered_listings if l.seller_id == seller_id
  ]

    if item_state:
        filtered_listings = [l for l in filtered_listings if l.item_state == item_sta
  te]

    return filtered_listings[skip:skip + limit]
```

Catatan:

• ✓ Endpoint read tetap public (tidak berubah setelah JWT)

• ✓ Mendukung filtering berdasarkan seller_id dan item_state

• ✓ Mendukung pagination dengan skip dan limit

Kelemahan Security Sebelum JWT

1. **Tidak Ada Autentikasi**

• Tidak ada cara untuk memverifikasi identitas user

• Siapa saja bisa mengklaim sebagai seller tertentu

2. **Tidak Ada Otorisasi**

• Tidak ada pengecekan kepemilikan

• User A bisa mengubah/menghapus listing milik User B

3. **Tidak Ada Session Management**

• Tidak ada tracking siapa yang sedang login

• Tidak ada token atau session identifier

4. **Rentan Terhadap Abuse**

• Mudah untuk spam create listing

• Mudah untuk sabotase listing orang lain

• Tidak ada rate limiting atau access control

Request/Response Example

Create Listing Request

```
{
  "seller_id": "123e4567-e89b-12d3-a456-426614174000",
  "title": "Kamera Vintage Canon",
  "price": {
    "amount": 2999999,
    "currency": "IDR"
  },
  "condition": {
    "score": 8,
    "detailed_description": "Kondisi baik dengan sedikit bekas pakai",
    "known_defects": ["Goresan kecil di tutup lensa"]
  },
  "attributes": [
    {
      "name": "Brand",
      "value": "Canon"
    },
```

```
    {
      "name": "Tahun",
      "value": "1985"
    }
  ]
}
```

Response

```
{
  "listing_id": "987fcdeb-51a2-43e7-9abc-123456789def",
  "seller_id": "123e4567-e89b-12d3-a456-426614174000",
  "title": "Kamera Vintage Canon",
  "item_state": "Active",
  "price": {
    "amount": 2999999,
    "currency": "IDR"
  },
  "condition": {
    "score": 8,
    "detailed_description": "Kondisi baik dengan sedikit bekas pakai",
    "known_defects": ["Goresan kecil di tutup lensa"]
  },
  "attributes": [
    {
      "attribute_id": "abc123...",
      "name": "Brand",
      "value": "Canon"
    },
    {
      "attribute_id": "def456...",
      "name": "Tahun",
      "value": "1985"
    }
  ],
  "created_at": "2025-12-01T10:30:00",
  "updated_at": "2025-12-01T10:30:00"
}
```

Kesimpulan

API versi sebelum JWT authentication memiliki fungsionalitas CRUD yang lengkap dan mengikuti prinsip Domain-Driven Design, namun **tidak memiliki keamanan sama sekali**. Siapa saja bisa melakukan operasi apa saja tanpa verifikasi identitas atau kepemilikan.

Kebutuhan untuk JWT Authentication:

- ✓ Verifikasi identitas user
- ✓ Ownership check untuk operasi sensitive
- ✓ Session management dengan token
- ✓ Authorization berbasis role/ownership