Dokumentasi API Setelah JWT Authentication

Gambaran Umum

API Item Management sekarang dilengkapi dengan JWT (JSON Web Token) authentication untuk mengamankan endpoints dan memastikan hanya user yang terautentikasi yang dapat melakukan operasi tertentu.

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
pyjwt==2.8.0
passlib[bcrypt]==1.7.4
```

Install packages:

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

Verifikasi instalasi:

```
python -c "import fastapi; print('FastAPI:', fastapi.__version__)"
python -c "import jwt; print('PyJWT:', jwt.__version__)"
python -c "import passlib; print('Passlib: OK')"
```

Langkah 3: Jalankan API Server

Menggunakan Python:

```
python listing_api.py
```

Atau menggunakan Uvicorn:

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

Output yang diharapkan:

```
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process
INFO:     Started server process
INFO:     Application startup complete.
```

Akses API Documentation:

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

• ReDoc: http://localhost:8000/redoc

Langkah 4: Testing API

Register User:

```
curl -X POST "http://localhost:8000/register" ^
  -H "Content-Type: application/json" ^
  -d "{\"username\": \"johndoe\", \"password\": \"password123\"}"
```

Login:

```
curl -X POST "http://localhost:8000/login" ^
  -H "Content-Type: application/json" ^
  -d "{\"username\": \"johndoe\", \"password\": \"password123\"}"
```

Create Listing (dengan token):

```
curl -X POST "http://localhost:8000/listings" ^
  -H "Authorization: Bearer YOUR_TOKEN" ^
  -H "Content-Type: application/json" ^
  -d "{\"title\": \"Laptop Gaming\", ...}"
```

Perubahan Struktur Kode

1. Import Dependencies Baru

```
from fastapi import FastAPI, HTTPException, status, Depends
from fastapi.security import HTTPBearer, HTTPAuthorizationCredentials
import jwt
from passlib.context import CryptContext
from datetime import datetime, timedelta
```

Penjelasan Tambahan:

• `Depends`: Untuk dependency injection (autentikasi)

• `HTTPBearer`: Security scheme untuk Bearer token

• `jwt`: Library untuk encode/decode JWT tokens

• `CryptContext`: Untuk hashing password dengan bcrypt

• `timedelta`: Untuk set waktu expiry token

2. Security Configuration

```
# Security Configuration
SECRET_KEY = "your-secret-key-change-in-production-min-32-chars-long"
ALGORITHM = "HS256"
ACCESS_TOKEN_EXPIRE_MINUTES = 30

security = HTTPBearer()
pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")
```

Penjelasan:

• `SECRET_KEY`: Kunci rahasia untuk sign JWT token (HARUS diganti di production!)

• `ALGORITHM`: Algoritma HS256 (HMAC + SHA256) untuk signing

• `ACCESS_TOKEN_EXPIRE_MINUTES`: Token expire dalam 30 menit

• `pwd_context`: Context untuk hash password dengan bcrypt

3. User Models

```
class User(BaseModel):
    user_id: UUID = Field(default_factory=uuid4)
    username: str
    hashed_password: str
    seller_id: UUID = Field(default_factory=uuid4)
```

```
class UserCreate(BaseModel):
    username: str = Field(..., min_length=3, max_length=50)
    password: str = Field(..., min_length=6)

class UserLogin(BaseModel):
    username: str
    password: str

class Token(BaseModel):
    access_token: str
    token_type: str

class TokenData(BaseModel):
    username: Optional[str] = None
```

Penjelasan:

• `User`: Model untuk menyimpan data user (password di-hash!)

• `UserCreate`: Request model untuk registrasi (password plain text)

• `UserLogin`: Request model untuk login

• `Token`: Response model berisi JWT token

• `TokenData`: Data yang di-decode dari JWT token

• Setiap user otomatis mendapat `seller_id` unik

4. Storage dengan User Database

```
listings_db: dict[UUID, Listing] = {}
search_index_db: dict[UUID, SearchIndexModel] = {}
users_db: dict[str, User] = {}  # BARU: Database user
```

Penjelasan:

• Ditambahkan `users_db` untuk menyimpan user credentials

• Key adalah username (unique)

• Value adalah object User dengan password yang sudah di-hash

5. Security Functions

Password Hashing

```
def verify_password(plain_password: str, hashed_password: str) -> bool:
    """Verify a password against its hash"""
    return pwd_context.verify(plain_password, hashed_password)

def get_password_hash(password: str) -> str:
    """Hash a password"""
    return pwd_context.hash(password)
```

Penjelasan:

• `verify_password`: Membandingkan password plain text dengan hash (untuk login)

• `get_password_hash`: Hash password menggunakan bcrypt (untuk registrasi)

• Bcrypt secara otomatis menambahkan salt untuk keamanan

JWT Token Management

```
def create_access_token(data: dict, expires_delta: Optional[timedelta] = None) -> str
  :
    """Create a JWT access token"""
    to_encode = data.copy()
    if expires_delta:
        expire = datetime.utcnow() + expires_delta
    else:
        expire = datetime.utcnow() + timedelta(minutes=15)
    to_encode.update({"exp": expire})
    encoded_jwt = jwt.encode(to_encode, SECRET_KEY, algorithm=ALGORITHM)
    return encoded_jwt
```

Penjelasan:

• Membuat JWT token dengan payload data (biasanya username)

• Menambahkan expiration time ke payload

• Sign dengan SECRET_KEY menggunakan algoritma HS256

• Return token string yang bisa dikirim ke client

Authentication Functions

```
def get_user(username: str) -> Optional[User]:
    """Get user from database"""
    return users_db.get(username)

def authenticate_user(username: str, password: str) -> Optional[User]:
    """Authenticate a user"""
    user = get_user(username)
    if not user:
        return None
    if not verify_password(password, user.hashed_password):
        return None
    return user
```

Penjelasan:

• `get_user`: Ambil user dari database berdasarkan username

• `authenticate_user`: Verifikasi username dan password

• Return user jika valid

• Return None jika invalid (username tidak ada atau password salah)

Current User Dependency

```
async def get_current_user(credentials: HTTPAuthorizationCredentials = Depends(securi
  ty)) -> User:
    """Get current user from JWT token"""
    credentials_exception = HTTPException(
        status_code=status.HTTP_401_UNAUTHORIZED,
        detail="Could not validate credentials",
        headers={"WWW-Authenticate": "Bearer"},
    )
    try:
        token = credentials.credentials
        payload = jwt.decode(token, SECRET_KEY, algorithms=[ALGORITHM])
        username: str = payload.get("sub")
        if username is None:
            raise credentials_exception
        token_data = TokenData(username=username)
    except jwt.PyJWTError:
        raise credentials_exception

    user = get_user(username=token_data.username)
    if user is None:
        raise credentials_exception
    return user
```

Penjelasan:

• Ini adalah dependency yang akan digunakan di endpoints

• Extract token dari Authorization header (Bearer token)

• Decode dan verify token menggunakan SECRET_KEY

• Extract username dari payload (field "sub")

• Load user dari database

• Raise 401 Unauthorized jika ada error

• Return user object jika valid

6. Endpoints Baru - Authentication

Register User

```
@app.post("/register", response_model=dict, status_code=status.HTTP_201_CREATED)
async def register_user(user_data: UserCreate):
    """Register a new user"""
    if user_data.username in users_db:
        raise HTTPException(
            status_code=status.HTTP_400_BAD_REQUEST,
            detail="Username already registered"
        )

    user = User(
        username=user_data.username,
        hashed_password=get_password_hash(user_data.password)
    )
    users_db[user.username] = user
```

```
    return {
        "message": "User registered successfully",
        "user_id": str(user.user_id),
        "seller_id": str(user.seller_id)
    }
```

Penjelasan:

• Endpoint public untuk registrasi user baru

• Check apakah username sudah ada (harus unique)

• Hash password sebelum disimpan (JANGAN simpan plain password!)

• Generate otomatis user_id dan seller_id

• Return info user yang baru dibuat

Login User

```
@app.post("/login", response_model=Token)
async def login(user_data: UserLogin):
    """Login and get access token"""
    user = authenticate_user(user_data.username, user_data.password)
    if not user:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Incorrect username or password",
            headers={"WWW-Authenticate": "Bearer"},
        )

    access_token_expires = timedelta(minutes=ACCESS_TOKEN_EXPIRE_MINUTES)
    access_token = create_access_token(
        data={"sub": user.username}, expires_delta=access_token_expires
    )

    return {"access_token": access_token, "token_type": "bearer"}
```

Penjelasan:

• Endpoint public untuk login

• Verifikasi username dan password

• Return 401 jika credentials salah

• Generate JWT token dengan expiry 30 menit

• Payload token berisi username di field "sub"

• Return token dan type "bearer"

7. Protected Endpoints

Create Listing (DENGAN AUTENTIKASI)

```
@app.post("/listings", response_model=ListingResponse, status_code=status.HTTP_201_CR
  EATED)
async def create_listing(
    request: CreateListingRequest,
    current_user: User = Depends(get_current_user)  # BARU!
):
    """Create a new listing (requires authentication)"""
    # Use the authenticated user's seller_id
    listing = Listing(
        seller_id=current_user.seller_id,  # BARU: Dari user yang login
        title=request.title,
        price=request.price,
        condition=request.condition,
        attributes=request.attributes or []
    )

    listings_db[listing.listing_id] = listing

    # Create search index
    search_index = SearchIndexModel(
        index_id=listing.listing_id,
        searchable_text="",
        relevance_score=0.0
    )
    search_index.update_from_listing(listing)
    search_index_db[listing.listing_id] = search_index

    # Publish event
    event = listing.publish_listing_indexed_event()

    return listing
```

Perubahan:

- ✓ Menambahkan `current_user: User = Depends(get_current_user)`
- ✓ Otomatis verifikasi JWT token
- ✓ Seller ID diambil dari user yang login, BUKAN dari request
- ✓ User tidak bisa mengaku sebagai seller lain

Update Price (DENGAN AUTHORIZATION)

```
@app.patch("/listings/{listing_id}/price", response_model=ListingResponse)
async def update_listing_price(
    listing_id: UUID,
    request: UpdatePriceRequest,
    current_user: User = Depends(get_current_user)  # BARU!
):
    """Update listing price (requires authentication)"""
    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]

    # Check if user owns this listing - BARU!
    if listing.seller_id != current_user.seller_id:
        raise HTTPException(
            status_code=status.HTTP_403_FORBIDDEN,
            detail="You can only update price of your own listings"
```

```
        )

    try:
        listing.update_price(request.new_price)
    except ValueError as e:
        raise HTTPException(
            status_code=status.HTTP_400_BAD_REQUEST,
            detail=str(e)
        )

    # Update search index
    if listing_id in search_index_db:
        search_index_db[listing_id].update_from_listing(listing)

    return listing
```

Perubahan:

• ✓ Menambahkan autentikasi dengan dependency

• ✓ **Ownership check**: Verifikasi apakah listing milik user ini

• ✓ Return 403 Forbidden jika user mencoba update listing orang lain

• ✓ Hanya owner yang bisa update harga

Delete Listing (DENGAN AUTHORIZATION)

```
@app.delete("/listings/{listing_id}", status_code=status.HTTP_204_NO_CONTENT)
async def delete_listing(
    listing_id: UUID,
    current_user: User = Depends(get_current_user)  # BARU!
):
    """Delete a listing (requires authentication)"""
    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]

    # Check if user owns this listing - BARU!
    if listing.seller_id != current_user.seller_id:
        raise HTTPException(
            status_code=status.HTTP_403_FORBIDDEN,
            detail="You can only delete your own listings"
        )

    del listings_db[listing_id]
    if listing_id in search_index_db:
        del search_index_db[listing_id]
```

Perubahan:

• ✓ Menambahkan autentikasi

• ✓ Ownership check sebelum delete

• ✓ Return 403 jika bukan owner

8. Activate & Delist Listings

Kedua endpoint ini juga ditambahkan authentication dan ownership check dengan pola yang sama:

```
@app.patch("/listings/{listing_id}/activate", response_model=ListingResponse)
async def activate_listing(
    listing_id: UUID,
    current_user: User = Depends(get_current_user)
):
    # ... ownership check ...

@app.patch("/listings/{listing_id}/delist", response_model=ListingResponse)
async def delist_listing(
    listing_id: UUID,
    request: DelistRequest,
    current_user: User = Depends(get_current_user)
):
    # ... ownership check ...
```

9. Public Endpoints (Tidak Berubah)

Endpoints berikut tetap public dan tidak memerlukan autentikasi:

- `GET /listings/{listing_id}` - Lihat detail listing
- `GET /listings` - List semua listing dengan filter
- `GET /search` - Pencarian listing
- `GET /health` - Health check

Flow Autentikasi

1. Registrasi

```
Client -> POST /register
       -> {username, password}

Server -> Hash password dengan bcrypt
       -> Simpan user ke database
       -> Generate user_id dan seller_id

Server -> Client: {message, user_id, seller_id}
```

2. Login

```
Client -> POST /login
       -> {username, password}

Server -> Cari user di database
       -> Verify password dengan bcrypt
```

```
        -> Generate JWT token (expire 30 menit)

Server -> Client: {access_token, token_type: "bearer"}
```

3. Akses Protected Endpoint

```
Client -> POST /listings
       -> Header: Authorization: Bearer <jwt_token>
       -> Body: {title, price, condition, ...}

Server -> Extract token dari header
       -> Decode dan verify token
       -> Load user dari database
       -> Execute business logic dengan user context

Server -> Client: Created listing dengan seller_id dari user
```

HTTP Status Codes

- **200 OK**: Request berhasil
- **201 Created**: Resource baru berhasil dibuat
- **204 No Content**: Delete berhasil
- **400 Bad Request**: Request tidak valid (misal: username sudah ada)
- **401 Unauthorized**: Token tidak valid atau tidak ada
- **403 Forbidden**: User tidak memiliki permission (bukan owner)
- **404 Not Found**: Resource tidak ditemukan

Security Features

1. Password Security

- ✓ Password di-hash dengan bcrypt
- ✓ Bcrypt otomatis menambahkan salt
- ✓ Password plain text tidak pernah disimpan
- ✓ Minimum password length: 6 karakter

2. Token Security

- ✓ JWT signed dengan SECRET_KEY
- ✓ Token expire setelah 30 menit

• ✓ Token berisi minimal info (hanya username)
• ✓ Token diverifikasi di setiap protected request

3. Authorization

• ✓ Ownership check untuk semua operasi modifikasi
• ✓ User hanya bisa modify/delete listing milik sendiri
• ✓ Return 403 Forbidden untuk unauthorized access

4. Input Validation

• ✓ Username: 3-50 karakter
• ✓ Password: minimum 6 karakter
• ✓ Pydantic validation untuk semua input

Keuntungan JWT Authentication

1. **Stateless**: Server tidak perlu menyimpan session
2. **Scalable**: Mudah di-scale horizontal
3. **Cross-domain**: Bisa digunakan di multiple services
4. **Self-contained**: Token berisi semua info yang dibutuhkan
5. **Standard**: JWT adalah industry standard

Kesimpulan

Dengan implementasi JWT authentication:
• ✓ API sekarang aman dari unauthorized access
• ✓ Setiap operasi diverifikasi kepemilikannya
• ✓ Password tersimpan dengan aman (bcrypt hash)
• ✓ Token management otomatis dengan expiry
• ✓ Clear separation antara public dan protected endpoints
• ✓ User tidak bisa memanipulasi seller_id orang lain