Dokumentasi Deployment, Testing, dan CI/CD

TST-API-DDD: Item Management API

1. Deployment dengan Render

1.1 Persiapan Deployment

Proyek ini telah di-deploy menggunakan platform **Render** yang menyediakan hosting gratis untuk aplikasi web dan API.

URL Deployment: https://api-tst.onrender.com

1.2 Langkah-langkah Deployment

Step 1: Persiapan File Konfigurasi

File-file yang diperlukan untuk deployment:

1. **requirements.txt** - Daftar dependencies Python:

```
fastapi==0.115.5
uvicorn[standard]==0.34.0
pydantic==2.10.5
pyjwt==2.10.1
passlib[bcrypt]==1.7.4
bcrypt==4.0.1
```

2. **listing_api.py** - File aplikasi utama dengan FastAPI

Step 2: Konfigurasi Render

1. Login ke [Render.com](https://render.com)

2. Klik "New +" → "Web Service"

3. Connect GitHub repository: `DoItMark/TST-API-DDD`

4. Konfigurasi service:

- **Name:** api-tst

- **Region:** Singapore (untuk latency lebih baik di Indonesia)

- **Branch:** main
- **Runtime:** Python 3
- **Build Command:** `pip install -r requirements.txt`
- **Start Command:** `uvicorn listing_api:app --host 0.0.0.0 --port $PORT`

Step 3: Environment Variables

Render secara otomatis menyediakan:
- `PORT` - Port yang akan digunakan aplikasi
- Public URL otomatis ter-generate

Step 4: Deploy

1. Klik "Create Web Service"
2. Render akan otomatis:
- Clone repository
- Install dependencies
- Start aplikasi
- Assign public URL

1.3 Karakteristik Deployment

Kelebihan Render:
- ■ Free tier tersedia
- ■ Auto-deploy saat push ke GitHub
- ■ HTTPS otomatis (SSL/TLS)
- ■ Health checks otomatis
- ■ Log monitoring

Limitasi Free Tier:
- ■■ Service sleep setelah 15 menit tidak ada aktivitas
- ■■ Cold start ~30 detik saat bangun dari sleep
- ■■ 750 jam compute per bulan
- ■■ Bandwidth terbatas

1.4 Monitoring Deployment

Health Check Endpoint:

```
GET https://api-tst.onrender.com/health
```

Response:

```
{
  "status": "healthy"
}
```

2. Testing Strategy

2.1 Overview Testing

Proyek ini mengimplementasikan **Test-Driven Development (TDD)** dengan coverage **94%**.

Total Tests: 44 test cases

Framework: pytest, pytest-cov, pytest-asyncio

Test Coverage: 94% (247/264 statements)

2.2 Struktur Testing

2.2.1 Test Classes dan Cakupan

1. **TestAuthentication (8 tests)**

- Registrasi user baru

- Login dengan kredensial yang benar/salah

- Validasi JWT token

- Akses endpoint protected dengan/tanpa token

2. **TestListingOperations (11 tests)**

- CRUD operations (Create, Read, Update, Delete)

- Filter listings berdasarkan seller

- Update harga listing

- Aktivasi dan delist listing

- Validasi ownership

3. **TestSecurity (8 tests)**

- Unauthorized access handling

- Ownership validation

- Non-owner operation failures

- 401/403 status codes

4. **TestSearchFunctionality (5 tests)**

- Search by query

- Relevance filtering

- Empty query handling

- No results handling

- Pagination

5. **TestHealthCheck (1 test)**

- Health endpoint status

6. **TestEdgeCases (11 tests)**

- Minimal data handling

- State transitions (DELISTED ↔ ACTIVE)

- Pagination

- Listings with attributes

- Known defects handling

- Relevance sorting

2.3 Test Fixtures

Fixtures yang digunakan:

```
@pytest.fixture
def client():
    """Test client untuk API"""
    return TestClient(app)

@pytest.fixture
def clear_databases():
    """Clear semua data sebelum test"""
    users_db.clear()
    listings_db.clear()
    search_index.clear()

@pytest.fixture
def sample_user_data():
    """Data user untuk testing"""
    return {
        "username": "testuser",
        "password": "testpass123",
        "full_name": "Test User"
    }
```

```
@pytest.fixture
def registered_user(client, clear_databases, sample_user_data):
    """User yang sudah terdaftar"""
    response = client.post("/register", json=sample_user_data)
    return sample_user_data

@pytest.fixture
def authenticated_user(client, registered_user):
    """User dengan token JWT"""
    response = client.post("/login", data={
        "username": registered_user["username"],
        "password": registered_user["password"]
    })
    token = response.json()["access_token"]
    return {"user": registered_user, "token": token}
```

2.4 Coverage Report

Coverage Details:

```
Name             Stmts   Miss  Cover   Missing
------------------------------------------------
listing_api.py     264     16    94%
------------------------------------------------
TOTAL              264     16    94%
```

Uncovered Lines (16 statements):

- Line 24-25: Import guards

- Line 87: Error handling edge case

- Line 135: Specific state validation

- Line 150: Domain event publishing

- Line 253: Token expiry edge case

- Lines 286, 293, 415, 431-432, 448, 481, 497-498, 517: Error responses dan edge cases

2.5 Menjalankan Tests Lokal

Prerequisites:

```
# Install dependencies
pip install -r requirements.txt
pip install -r requirements-dev.txt
```

Run All Tests:

```
pytest -v
```

Run dengan Coverage:

```
pytest --cov=listing_api --cov-report=term-missing -v
```

Run dengan Coverage HTML:

```
pytest --cov=listing_api --cov-report=html -v
# Buka htmlcov/index.html di browser
```

Run Specific Test Class:

```
pytest test_listing_api.py::TestAuthentication -v
```

2.6 Test Results

Hasil Running Tests:

```
====== test session starts ======
platform win32 -- Python 3.14.0, pytest-8.3.4
collected 44 items

TestAuthentication (8 tests)     ✓ PASSED
TestListingOperations (11 tests) ✓ PASSED
TestSecurity (8 tests)           ✓ PASSED
TestSearchFunctionality (5 tests)✓ PASSED
TestHealthCheck (1 test)         ✓ PASSED
TestEdgeCases (11 tests)         ✓ PASSED

====== 44 passed in 17.87s ======
```

3. CI/CD Pipeline

3.1 Overview CI/CD

Proyek ini menggunakan **GitHub Actions** untuk automation testing dan deployment.

Pipeline File: `.github/workflows/ci.yml`

3.2 Konfigurasi Pipeline

3.2.1 Trigger Events

Pipeline otomatis berjalan pada:

```
on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]
```

3.2.2 Multi-Version Testing

Testing dilakukan pada multiple Python versions:

```
    strategy:
      matrix:
        python-version: ['3.11', '3.12', '3.13']
```

Alasan:

- Memastikan compatibility across Python versions

- Deteksi breaking changes lebih awal

- Future-proofing code

3.3 Pipeline Steps

Step 1: Checkout Code

```
    - uses: actions/checkout@v4
```

Clone repository ke runner

Step 2: Setup Python

```
    - name: Set up Python ${{ matrix.python-version }}
      uses: actions/setup-python@v5
      with:
        python-version: ${{ matrix.python-version }}
```

Install Python version yang ditentukan

Step 3: Cache Dependencies

```
    - name: Cache pip packages
      uses: actions/cache@v3
      with:
        path: ~/.cache/pip
        key: ${{ runner.os }}-pip-${{ hashFiles('**/requirements*.txt') }}
```

Manfaat:

- Faster builds (tidak download ulang packages)

- Reduce bandwidth usage

- Konsisten dependencies

Step 4: Install Dependencies

```
    - name: Install dependencies
      run: |
        python -m pip install --upgrade pip
        pip install -r requirements.txt
        pip install -r requirements-dev.txt
```

Step 5: Run Tests dengan Coverage

```
- name: Run tests with coverage
  run: |
    pytest --cov=listing_api --cov-report=term-missing --cov-report=xml --cov-report=html -v
```

Output:

- Terminal output untuk quick review

- XML report untuk CI integration

- HTML report untuk detailed analysis

Step 6: Coverage Threshold Check

```
- name: Coverage threshold check
  run: |
    coverage report --fail-under=80
```

Enforce minimum 80% coverage

Step 7: Upload Artifacts

```
- name: Upload coverage reports
  uses: actions/upload-artifact@v4
  with:
    name: coverage-report-py${{ matrix.python-version }}
    path: |
      htmlcov/
      coverage.xml
```

3.4 Pipeline Success Criteria

Pipeline dianggap **SUCCESS** jika:

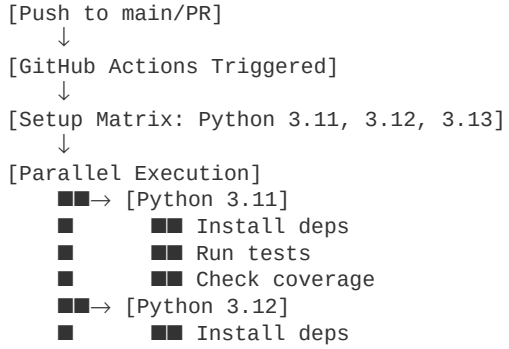
1. ■ All tests pass (44/44)

2. ■ Coverage ≥ 80% (saat ini: 94%)

3. ■ No critical errors

4. ■ Berjalan pada semua Python versions (3.11, 3.12, 3.13)

3.5 Workflow Visualization



```
    ■      ■■ Run tests
    ■      ■■ Check coverage
    ■■→ [Python 3.13]
           ■■ Install deps
           ■■ Run tests
           ■■ Check coverage
    ↓
[All Jobs Success?]
    ■■ YES → ■ Merge Allowed / Auto-deploy
    ■■ NO  → ■ Block Merge / Alert
```

3.6 Integration dengan Render

Auto-Deploy Flow:

```
[Developer]
    ↓
[git push origin main]
    ↓
[GitHub Actions CI]
    ■■ Run tests
    ■■ Check coverage
    ■■ ■ All pass
    ↓
[GitHub main branch updated]
    ↓
[Render Auto-Deploy Triggered]
    ■■ Pull latest code
    ■■ Install dependencies
    ■■ Restart service
    ■■ ■ Live at https://api-tst.onrender.com
```

3.7 Monitoring CI/CD

GitHub Actions Dashboard:

- View runs: https://github.com/DoItMark/TST-API-DDD/actions

- Status badges dapat ditambahkan ke README

- Email notifications untuk failures

Best Practices Implemented:

1. ■ Fast feedback (17s test execution)

2. ■ Parallel execution (3 Python versions)

3. ■ Dependency caching

4. ■ Artifact preservation

5. ■ Coverage enforcement

6. ■ Multi-version compatibility

4. Architecture dan Design Patterns

4.1 Domain-Driven Design (DDD)

Aggregate Root:

- `Listing` - Root entity yang mengelola lifecycle listing

Value Objects:

- `Money` - Immutable price dengan currency
- `Condition` - Kondisi item (NEW/USED/REFURBISHED)
- `DelistReason` - Alasan delist dengan notes
- `Filter` - Search filtering parameters

Entities:

- `Attribute` - Custom attributes untuk listing

Domain Events:

- `ListingIndexedEvent` - Event saat listing di-index

Read Model:

- `SearchIndexModel` - Optimized untuk search queries

4.2 API Security

Authentication:

- JWT (JSON Web Token) dengan expiry 30 menit
- Password hashing menggunakan bcrypt
- Bearer token authentication scheme

Authorization:

- Ownership validation untuk operations
- Role-based access (seller-based)

5. Kesimpulan

5.1 Summary

Proyek TST-API-DDD berhasil mengimplementasikan:

1. ■ **RESTful API** dengan FastAPI
2. ■ **Domain-Driven Design** principles
3. ■ **JWT Authentication** dan authorization
4. ■ **Comprehensive Testing** (94% coverage, 44 tests)
5. ■ **CI/CD Pipeline** dengan GitHub Actions
6. ■ **Production Deployment** di Render.com

5.2 Metrics

Metric	Value
Test Coverage	94%
Total Tests	44
API Endpoints	11
Python Versions Tested	3 (3.11, 3.12, 3.13)
CI Pipeline Duration	~2-3 menit
Test Execution Time	17.87s
Deployment Platform	Render.com
Uptime	24/7 (dengan cold start)

5.3 Future Improvements

Potential Enhancements:

1. Database persistence (PostgreSQL/MongoDB)
2. Rate limiting untuk API endpoints
3. API versioning (v1, v2)
4. Swagger UI customization
5. Monitoring dan logging (Sentry, LogDNA)
6. Performance testing (Locust, JMeter)
7. Security scanning (Bandit, Safety)
8. Code quality checks (Black, Flake8)

Dibuat oleh: Mark (18223130)

Mata Kuliah: II3160 - Teknologi Sistem Terintegrasi

Tanggal: 4 Januari 2026

■■ CATATAN PENTING - AI GENERATION

Dokumentasi ini dibuat dengan bantuan GitHub Copilot (AI Assistant).

Sebagian besar konten dalam dokumen ini di-generate menggunakan GitHub Copilot untuk:

- Mempercepat proses dokumentasi teknis
- Memastikan kelengkapan informasi deployment, testing, dan CI/CD
- Menjaga konsistensi format dan struktur dokumentasi
- Memberikan penjelasan yang komprehensif dan mudah dipahami

Kontribusi Manual:

- Konfigurasi deployment di Render.com
- Implementasi test suite dan CI/CD pipeline
- Validasi teknis semua instruksi dan command
- Review dan verifikasi akurasi konten

Tools yang digunakan:

- GitHub Copilot untuk AI-assisted content generation
- ReportLab untuk konversi Markdown ke PDF
- Manual review untuk quality assurance

Dokumentasi ini merupakan bagian dari tugas kuliah II3160 (Teknologi Sistem Terintegrasi) yang mengimplementasikan best practices dalam software engineering dengan bantuan AI tools modern.