

**Deliverable 6**  
**FINALISASI**  
**II3160 - Integrated Systems Technology**



**Disusun oleh:**  
Theresia Ivana Marella Siswahyudi - 18223126

**Program Studi Sistem Dan Teknologi Informasi**  
**Sekolah Teknik Elektro Dan Informatika**  
**Institut Teknologi Bandung**  
**2025**

# DAFTAR ISI

<b>DAFTAR ISI.....</b>	<b>2</b>
<b>LATAR BELAKANG.....</b>	<b>4</b>
<b>DESAIN ARSITEKTUR IMPLEMENTASI.....</b>	<b>5</b>
<b>IMPLEMENTASI SISTEM AUTENTIKASI &amp; OTORISASI BERBASIS JWT.....</b>	<b>7</b>
A. Konsep Autentikasi JWT .....	7
B. Komponen Autentikasi.....	7
1) Registrasi Pengguna (/auth/register).....	7
2) Login (/auth/login).....	8
3) Logout (/auth/logout).....	8
C. Struktur Payload Token.....	8
E. Mekanisme Validasi Token.....	9
F. Token Blacklist.....	9
G. Integrasi Autentikasi dengan API Layer.....	9
<b>IMPLEMENTASI DATABASE &amp; MIGRASI REPOSITORY.....</b>	<b>11</b>
A. Model Data dan Struktur Tabel.....	11
1) Tabel users.....	11
2) Tabel inventory_items.....	11
3) Tabel reservations.....	11
B. Repository Baru: InventoryRepositoryDB.....	11
C. Konversi Domain Model dan Model Database.....	12
D. Integrasi Repository DB dengan Application Layer.....	12
<b>IMPLEMENTASI DOMAIN MODEL.....</b>	<b>13</b>
A. Aggregate Root: InventoryItem.....	13
B. Value Objects.....	13
C. Entities.....	13
<b>IMPLEMENTASI APPLICATION LAYER.....</b>	<b>15</b>
A. Repository: InventoryRepositoryInMemory.....	15
B. Service: InventoryService.....	15
<b>IMPLEMENTASI API LAYER.....</b>	<b>17</b>
A. Struktur Umum API.....	17
B. Endpoint Admin Gudang (/admin/...).....	17
C. Endpoint Open Host Service (/ohs/...).....	18
D. Endpoint Manager (/manager/...).....	18
E. Integrasi dengan Pydantic Schema.....	19
F. Integrasi Autentikasi & Role-Based Access Control.....	19
<b>UNIT TESTING DAN TEST DRIVEN DEVELOPMENT (TDD).....</b>	<b>20</b>
A. Konsep TDD dan Test Coverage.....	20
B. Struktur Unit Test.....	20
C. Contoh Test Cases.....	21
D. Hasil Coverage.....	22
<b>CI/CD DENGAN GITHUB WORKFLOWS.....</b>	<b>23</b>
A. Struktur Workflow.....	24
B. Tahapan Workflow.....	24
C. Manfaat Integrasi Docker dalam CI/CD.....	26

<b>DOKUMENTASI REQUEST &amp; RESPONSE PAYLOAD.....</b>	<b>27</b>
A. Authentication Endpoints.....	27
B. Health & Status.....	28
C. Admin Endpoints (Inventory Management).....	28
D. Client/OHS Endpoints (Stock Operations).....	32
E. Manager Endpoints (Monitoring).....	35
<b>MENJALANKAN APLIKASI DAN LINGKUNGAN EKSEKUSI.....</b>	<b>37</b>
1. Instalasi dan Persiapan Lingkungan.....	37
2. Menjalankan Aplikasi.....	37
3. Dokumentasi & Pengujian API.....	38
<b>PENGUJIAN API DENGAN SWAGGER UI &amp; POSTMAN.....</b>	<b>40</b>
A. Pengujian Menggunakan Swagger UI.....	41
B. Pengujian Menggunakan Postman.....	67
C. Pengujian Menggunakan pytest.....	76
D. Kesimpulan Hasil Pengujian.....	78
<b>ACTIVITY DIAGRAM INVENTORY CONTROL SERVICE.....</b>	<b>79</b>
<b>KESIMPULAN.....</b>	<b>81</b>

## LATAR BELAKANG

Pengembangan modul *Inventory Control* sejak *deliverable 1* hingga *deliverable 4* telah membentuk landasan yang kuat bagi sistem manajemen pergudangan (*Warehouse Management System / WMS*). Tahap analisis awal pada *deliverable 1* mengidentifikasi kebutuhan bisnis serta proses inti yang harus didukung oleh sistem. *Deliverable 2* kemudian menetapkan batasan arsitektur melalui pemetaan *bounded context*, di mana *Inventory Control* ditetapkan sebagai *core context* yang menjadi sumber kebenaran data stok. Model domain untuk konteks ini dirumuskan secara rinci pada *deliverable 3*, dan pada *deliverable 4* model tersebut diwujudkan dalam implementasi awal menggunakan FastAPI, lengkap dengan aplikasi layanan, logika domain, dan *in-memory repository* untuk mendukung pengujian dan validasi fungsional.

Walaupun implementasi pada tahap sebelumnya telah mampu menjalankan operasi dasar pengelolaan stok, mulai dari penambahan dan pengurangan stok, reservasi barang, hingga pemantauan kondisi low stock, arsitektur yang digunakan masih berada pada tahap *prototype*. Sistem belum memiliki mekanisme autentikasi, sehingga seluruh *endpoint* dapat diakses tanpa pembatasan, dan penyimpanan data masih bergantung pada *in-memory repository*, yang hanya bersifat sementara dan tidak memenuhi kebutuhan persistensi data dalam penggunaan jangka panjang. Dalam ekosistem WMS yang melibatkan banyak aktor dan modul, kedua aspek ini menjadi sangat penting terutama karena *Inventory Control* berperan sebagai *Open Host Service* yang menyediakan data stok bagi konteks lain.

Pada *deliverable 5* dilanjutkan proses pengembangan sistem dengan memperkenalkan dua peningkatan utama. Pertama, penerapan autentikasi berbasis JSON Web Token (JWT) untuk memberikan kontrol akses yang lebih aman terhadap seluruh *endpoint Inventory Control*. Kedua, penggantian penyimpanan berbasis memori dengan database agar sistem dapat menyimpan data secara persisten, mendukung integritas informasi, serta membuka jalan bagi integrasi yang lebih luas di kemudian hari. Kedua peningkatan ini selaras dengan desain arsitektur yang telah dibangun sebelumnya, karena lapisan *repository* memang dirancang sebagai abstraksi yang memungkinkan pergantian mekanisme penyimpanan tanpa mengubah logika domain.

*Deliverable 6* memfokuskan pengembangan pada penjaminan kualitas perangkat lunak melalui pengujian unit (*unit testing*) dan penerapan praktik *Test-Driven Development* (TDD). Langkah ini dilakukan untuk memastikan bahwa seluruh fungsi *Inventory Control* berjalan sesuai spesifikasi, termasuk logika domain untuk pengelolaan stok, validasi payload, serta interaksi dengan database melalui *repository*. Dengan mengintegrasikan unit test sejak awal pengembangan fitur baru, pengembang dapat lebih cepat mendeteksi kesalahan, mengurangi risiko regresi, dan menjaga konsistensi perilaku sistem saat dilakukan perbaikan atau penambahan fitur. Pendekatan TDD juga mendorong desain kode yang lebih modular dan terstruktur, sehingga setiap komponen menjadi lebih mudah diuji secara terisolasi dan memudahkan pemeliharaan jangka panjang.

Selain itu, *deliverable 6* memperkenalkan Docker sebagai sarana untuk mengemas aplikasi beserta seluruh dependensinya dalam lingkungan container yang konsisten, sehingga memudahkan pengembangan, pengujian, dan deployment lintas lingkungan. *Continuous Integration* (CI) juga diterapkan untuk mengotomatiskan eksekusi *unit test* setiap kali ada perubahan kode, memastikan integritas sistem terjaga, dan mempercepat deteksi potensi error sebelum mencapai tahap produksi. Dengan kombinasi unit testing, TDD, Docker, dan CI, *deliverable 6* menjadi langkah strategis untuk meningkatkan stabilitas, keandalan, dan kesiapan operasional modul *Inventory Control* dalam konteks ekosistem WMS yang lebih luas.

## DESAIN ARSITEKTUR IMPLEMENTASI

Arsitektur implementasi modul Inventory Control dibangun dengan pendekatan berlapis yang mengacu pada prinsip Domain-Driven Design (DDD). Tujuannya adalah untuk memisahkan logika domain inti dari detail teknis implementasi, sehingga sistem menjadi lebih terstruktur, mudah dipahami, dan mudah dikembangkan pada tahap berikutnya. Pada tahap implementasi di *deliverable 5*, arsitektur yang sebelumnya terdiri dari *Domain Layer*, *Application Layer*, dan *API Layer* diperluas dengan dua lapisan tambahan, yaitu *Authentication Layer* dan *Infrastructure Layer* untuk *database*. Penambahan ini dilakukan untuk memenuhi kebutuhan keamanan dan persistensi data yang tidak termasuk pada implementasi awal.

Lapisan *Domain Layer* tetap menjadi pusat logika bisnis dan tidak mengalami perubahan dari *deliverable* sebelumnya. Lapisan ini berisi model inti yang merepresentasikan aturan pengelolaan stok melalui aggregate *InventoryItem*, beserta *Value Object* seperti *SKU*, *Quantity*, *Threshold*, dan *Batch*, serta *Entity* seperti *Reservation* dan *StockMove*. Seluruh aturan bisnis didefinisikan secara murni pada lapisan ini tanpa ketergantungan terhadap API, database, maupun detail teknis lainnya, sehingga tetap menjaga prinsip isolasi domain yang ditetapkan pada *deliverable 3*.

Lapisan *Application Layer* berfungsi sebagai penghubung antara domain dan antarmuka sistem. Pada *Deliverable 5*, lapisan ini tidak hanya mengkoordinasikan operasi seperti *increase*, *decrease*, *reserve*, dan *release*, tetapi juga berinteraksi dengan *repository* berbasis database melalui *InventoryRepositoryDB*. *Repository* ini menggantikan *InventoryRepositoryInMemory* yang digunakan pada *deliverable 4* dan menjadi komponen yang bertanggung jawab untuk memetakan objek domain ke dalam model *database*. Dengan tetap mempertahankan antarmuka yang sama, migrasi sistem penyimpanan dapat dilakukan tanpa mengubah struktur domain atau logika *use case*.

Lapisan *Infrastructure Layer* diperkenalkan pada tahap ini untuk menangani persistensi data menggunakan SQLAlchemy dan SQLite. Lapisan ini mencakup definisi tabel pengguna, item inventori, dan reservasi, serta menyediakan mekanisme koneksi database dan konversi antara model domain dan model tabel. Keberadaan lapisan ini memungkinkan sistem untuk menyimpan data secara persisten, memastikan integritas informasi, dan mendukung integrasi antar *bounded context* secara lebih stabil pada tahap pengembangan selanjutnya.

Lapisan baru berikutnya adalah *Authentication Layer*, yang memberikan mekanisme kontrol akses berbasis JSON Web Token (JWT). Lapisan ini menyediakan fungsi registrasi, *login*, *logout*, validasi token, dan otorisasi berbasis peran (*role-based access control*). Melalui lapisan ini, endpoint API dapat dilindungi sesuai peran masing-masing aktor, yaitu admin gudang, manager, dan client (sistem eksternal). Layer ini memastikan bahwa hanya pengguna yang berwenang yang dapat menjalankan operasi tertentu, sehingga mendukung keamanan dan keberlangsungan layanan *Inventory Control* sebagai *Open Host Service*.

Lapisan paling luar adalah *API Layer*, yang diimplementasikan menggunakan FastAPI. Pada *deliverable 5*, lapisan ini mengalami perluasan dengan memberikan proteksi *endpoint* berdasarkan *role* dan integrasi penuh dengan mekanisme autentikasi JWT. Endpoint tetap dikelompokkan sesuai kategorinya (admin, OHS, dan manager), namun kini setiap endpoint wajib membawa token akses dan diproses melalui dependency autentikasi dan otorisasi sebelum logika bisnis dijalankan. Selain itu, API Layer memanfaatkan Pydantic untuk validasi input dan penyusunan Data Transfer Object (DTO), memastikan bahwa data yang masuk ke sistem bersifat valid dan konsisten.

Dengan penambahan lapisan autentikasi dan database persistence, arsitektur implementasi pada *deliverable 5* tidak hanya mempertahankan struktur modular dari tahap sebelumnya, tetapi juga memberikan fondasi yang lebih matang dan siap dioperasikan dalam konteks sistem WMS yang sesungguhnya. Pendekatan berlapis ini menjamin bahwa sistem tetap fleksibel, aman, dan mudah diperluas pada pengembangan selanjutnya, baik untuk integrasi antarkonteks maupun kebutuhan operasional yang lebih kompleks.

**Tabel 1. Pemetaan Layer**

<b>Layer</b>	<b>Folder</b>	<b>Fungsi</b>
<b>Domain Layer</b>	src/domain/	Aturan bisnis inti, aggregate, entity, value object
<b>Application Layer</b>	src/services/	Use case, repository, koordinasi domain
<b>Infrastructure Layer</b>	src/db.py	ORM, model tabel SQLAlchemy, engine DB
<b>Auth Layer</b>	src/auth.py	Mekanisme login, register, logout, token validation
<b>API Layer</b>	src/main.py	Endpoint dilindungi JWT dan role-based access control

Renovasi arsitektur ini tetap menjaga prinsip DDD dan *separation of concerns*. Domain model tidak berubah sama sekali, sementara *repository* dan API diadaptasi agar mendukung autentikasi dan penyimpanan data persisten.

# IMPLEMENTASI SISTEM AUTENTIKASI & OTORISASI BERBASIS JWT

Penerapan autentikasi dan otorisasi pada *deliverable 5* merupakan peningkatan arsitektur yang penting untuk memastikan bahwa seluruh *endpoint Inventory Control* hanya dapat diakses oleh pengguna dan sistem yang berwenang. Mengingat modul ini berfungsi sebagai *Open Host Service* (OHS) bagi konteks lain seperti *Inbound*, *Outbound*, dan *Order Management*, maka mekanisme keamanan yang kokoh menjadi keharusan agar data stok tidak dapat dimanipulasi atau diakses sembarangan. Untuk mencapai hal tersebut, digunakan mekanisme autentikasi berbasis JSON Web Token (JWT) dan pengaturan hak akses berbasis peran (*Role-Based Access Control / RBAC*).

## A. Konsep Autentikasi JWT

JSON Web Token (JWT) adalah standar token berbasis JSON yang digunakan untuk autentikasi stateless. Pada implementasi ini, JWT memuat informasi penting seperti nama pengguna dan peran, kemudian ditandatangani menggunakan algoritma HMAC-SHA256. Karena JWT bersifat self-contained, validasi token dapat dilakukan tanpa menyimpan informasi sesi di server, sehingga arsitektur tetap ringan dan sesuai dengan pendekatan microservice maupun DDD. Struktur JWT terdiri dari tiga bagian:

```
<header>.⟨payload⟩.⟨signature⟩
```

Payload token pada sistem ini memuat:

- `sub` → username pengguna
- `role` → peran pengguna (admin/manager/client)
- `exp` → waktu kedaluwarsa token

Dengan adanya informasi ini, server dapat memverifikasi identitas pengguna dan menentukan apakah aksesnya sesuai dengan peran yang diberikan.

## B. Komponen Autentikasi

Autentikasi pada modul Inventory Control terdiri dari tiga proses utama:

### 1) Registrasi Pengguna (/auth/register)

Digunakan untuk membuat akun baru. Setiap pengguna memiliki atribut:

- `username`
- `full_name`
- `role`
- `hashed_password` (password disimpan menggunakan bcrypt)
- `disabled` (opsional)

Tabel berikut menunjukkan parameter utama registrasi:

Field	Tipe	Deskripsi
username	string	Identitas unik pengguna
password	string	Kata sandi yang akan di-hash
full_name	string	Nama lengkap pengguna
role	enum(admin, manager, client)	Menentukan hak akses

## 2) Login (/auth/login)

Pengguna memasukkan username dan password, kemudian server:

- Memverifikasi kecocokan password melalui bcrypt
- Menghasilkan token JWT jika autentikasi berhasil
- Mengembalikan response berupa:

```
{  
    "access_token": "<jwt_token>",  
    "token_type": "bearer"  
}
```

## 3) Logout (/auth/logout)

Logout dilakukan dengan cara:

- menambahkan token aktif ke dalam token blacklist
- setiap request berikutnya akan ditolak jika token berada di dalam daftar blacklist

Pendekatan ini memungkinkan mekanisme logout tetap aman meskipun JWT sendiri bersifat stateless.

## C. Struktur Payload Token

Payload token JWT pada implementasi ini adalah sebagai berikut:

```
{  
    "sub": "admin1",  
    "role": "admin",  
    "exp": 1732617599  
}
```

**Tabel 2. Struktur Payload Token**

Claim	Fungsi
sub	Identitas pengguna (username)
role	Menentukan hak akses pengguna
exp	Timestamp kedaluwarsa token

Dengan struktur ini, sistem dapat memverifikasi apakah token masih aktif, apakah token sah, dan apakah peran pengguna sesuai untuk *endpoint* tertentu.

## D. Role-Based Access Control (RBAC)

Pada *deliverable 5*, diterapkan sistem otorisasi berbasis peran yang menentukan *endpoint* apa saja yang dapat diakses oleh tiap jenis pengguna. Tabel berikut menjelaskan pembagian akses:

**Tabel 3. RBAC**

Role	Tujuan	Endpoint yang Diizinkan
admin	Mengelola master data inventori	/admin/...
manager	Memantau stok & laporan	/manager/...

<b>client</b>	Mengakses layanan OHS untuk inbound/outbound	/ohs/...
(tanpa token)	Tidak diizinkan	Semua endpoint ditolak

Implementasi teknis menggunakan dependency FastAPI:

```
Depends(require_role("admin"))
```

Kode di atas memvalidasi token, mengambil *payload*, memastikan `role == "admin"`, serta menolak *request* jika tidak sesuai.

## E. Mekanisme Validasi Token

Sebelum menjalankan logika bisnis, setiap endpoint terproteksi menjalankan proses validasi token yang mencakup:

1. Memastikan header Authorization berisi token dengan format:

```
Bearer <token>
```

2. Memverifikasi tanda tangan token menggunakan `SECRET_KEY`
3. Mengecek apakah token telah kedaluwarsa (`exp`)
4. Mengecek apakah token termasuk dalam blacklist
5. Mengecek apakah peran (`role`) sesuai dengan endpoint

Jika salah satu validasi gagal, sistem memberikan respons sesuai status:

- 401 → jika token invalid / expired
- 403 → jika role tidak memiliki izin
- 401 → jika token ada dalam blacklist (sudah logout)

## F. Token Blacklist

Meskipun JWT bersifat *stateless*, implementasi logout tetap memungkinkan melalui mekanisme *token blacklisting*. Ketika pengguna melakukan logout:

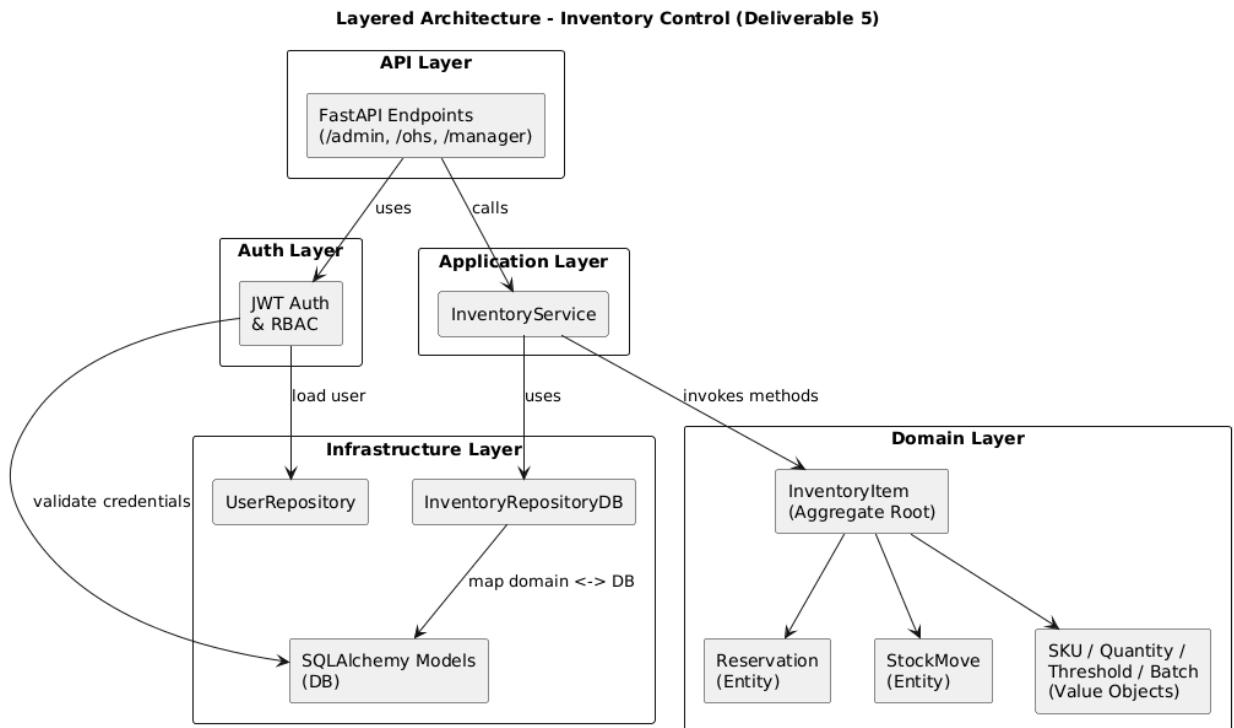
- token disimpan dalam set `TOKEN_BLACKLIST`
- pada setiap request berikutnya, token akan dicek apakah termasuk dalam daftar ini
- jika iya, request ditolak dengan pesan: “Logged out token”

Pendekatan ini memastikan pengguna tidak dapat menggunakan token lama setelah logout meskipun token tersebut belum kedaluwarsa.

## G. Integrasi Autentikasi dengan API Layer

Integrasi autentikasi dengan API *Layer* dilakukan dengan menambahkan mekanisme validasi token dan pemeriksaan peran pada setiap *endpoint Inventory Control*. Seluruh permintaan yang masuk kini diwajibkan untuk membawa token JWT yang sah dan belum berada dalam daftar token yang telah dinonaktifkan (*blacklist*). Setelah token diverifikasi, sistem juga memeriksa peran pengguna untuk menentukan apakah ia memiliki hak akses terhadap endpoint yang hendak digunakan. Misalnya, *endpoint* yang berada pada kelompok admin hanya dapat diakses oleh pengguna dengan peran admin, sementara *endpoint monitoring* pada kelompok manager dibatasi untuk peran manager, dan *endpoint Open Host Service (OHS)* dibuka khusus bagi peran client yang mewakili modul eksternal seperti *Inbound*, *Outbound*, atau *Order Management*.

Penerapan kontrol akses ini membuat API *Layer* berfungsi bukan sekadar sebagai antarmuka pemanggilan layanan, tetapi juga sebagai gerbang keamanan yang memastikan bahwa setiap operasi yang berhubungan dengan data stok dijalankan oleh pihak yang berwenang. Pendekatan ini menjadikan modul *Inventory Control* lebih aman, terstruktur, dan selaras dengan pembagian bounded context yang telah ditetapkan pada *deliverable 2*. Dengan demikian, integrasi autentikasi berbasis JWT tidak hanya memperkuat keamanan sistem, tetapi juga mendukung strategi arsitektur domain yang konsisten di seluruh tahapan pengembangan.



Gambar 0. *Layered Architecture Inventory Control* pada Deliverable 5

# IMPLEMENTASI DATABASE & MIGRASI REPOSITORY

Migrasi repository merupakan salah satu aspek utama dalam pengembangan pada *deliverable 5*. Pada tahap sebelumnya (*deliverable 4*), sistem *Inventory Control* menggunakan *repository* berbasis *in-memory* yang bertujuan untuk mendukung proses pengembangan awal dan pengujian logika domain. Meskipun pendekatan tersebut efektif untuk tahap *prototyping*, ia tidak memenuhi kebutuhan persistensi data, integritas informasi, maupun interoperabilitas jangka panjang. Oleh karena itu, pada *deliverable 5* dilakukan migrasi mekanisme penyimpanan ke *database* yang lebih stabil dan terstruktur menggunakan SQLAlchemy dan SQLite sebagai backend penyimpanan data.

## A. Model Data dan Struktur Tabel

Database diimplementasikan melalui modul db.py yang mendefinisikan struktur tabel menggunakan SQLAlchemy ORM. Tiga tabel utama disiapkan untuk mendukung operasi *Inventory Control* dan autentikasi:

### 1) Tabel users

Menyimpan informasi pengguna, termasuk username, nama lengkap, role, status akun, dan kata sandi yang telah di-hash menggunakan bcrypt. Tabel ini digunakan untuk autentikasi (login, register) dan proses otorisasi berdasarkan *role*.

### 2) Tabel inventory\_items

Merepresentasikan entitas *InventoryItem* pada domain model. Setiap baris menyimpan informasi SKU, stok fisik (on\_hand), stok yang telah dipesan (reserved), satuan barang (uom), dan ambang batas minimum sebelum dianggap low stock.

### 3) Tabel reservations

Menyimpan data reservasi yang dilakukan oleh modul pemesanan atau pengguna peran *client*. Tabel ini memuat hubungan antara SKU, jumlah yang direservasi, dan order ID untuk memastikan setiap reservasi tercatat secara eksplisit dan dapat ditelusuri.

Dengan adanya pemisahan tabel ini, sistem mampu mempertahankan integritas referensial dan mendukung pembuatan laporan atau integrasi modul lain pada tahap pengembangan berikutnya.

## B. Repository Baru: InventoryRepositoryDB

Untuk mempertahankan prinsip pemisahan tanggung jawab pada arsitektur DDD, mekanisme penyimpanan database tidak dihubungkan langsung ke domain model. Sebaliknya, *repositori database* diimplementasikan sebagai kelas `InventoryRepositoryDB` yang menyediakan antarmuka penyimpanan dengan fungsi yang setara dengan *repository in-memory* sebelumnya. Tujuannya adalah memastikan bahwa lapisan *Application Layer* tetap dapat menggunakan *repository* melalui kontrak yang sama meskipun backend penyimpanan berubah. Repository ini menyediakan fungsi utama seperti:

- `list_all()` → mengambil seluruh item inventori dari database
- `get_by_sku(sku)` → mengambil entitas domain berdasarkan SKU tertentu
- `save(item)` → menyimpan atau memperbarui data item di database

Proses penyimpanan dilakukan melalui mekanisme *mapping Domain → Model Database → Database*, menggunakan fungsi `domain_to_model` dan `model_to_domain` agar logika domain tetap

terjaga tanpa perubahan. Migrasi *repository* menjadi langkah penting yang memungkinkan sistem beralih dari penyimpanan sementara menuju penyimpanan persisten tanpa mempengaruhi struktur dan perilaku domain model.

### C. Konversi Domain Model dan Model Database

Pemetaan dua arah antara domain model dan database diterapkan agar domain model tetap bersifat murni dan tidak bergantung pada detail teknis representasi data. Proses pemetaan dilakukan melalui dua fungsi utama:

- `model_to_domain()` → Mengubah baris tabel menjadi objek domain (InventoryItem, SKU, Quantity, Threshold).
- `domain_to_model()` → Mengonversi objek domain kembali ke objek database ORM sebelum disimpan ke SQLAlchemy.

Dengan pendekatan ini, domain model tetap berfokus pada aturan bisnis dan tidak terikat pada database tertentu. Hal ini selaras dengan prinsip DDD bahwa domain harus berada pada inti arsitektur dan tidak dipengaruhi oleh teknologi penyimpanan.

### D. Integrasi Repository DB dengan Application Layer

Lapisan *Application Layer* pada *deliverable 5* memanfaatkan *repository* berbasis database melalui konstruktor `InventoryService`. Perubahan *repository* tidak mengubah cara kerja *service* karena antarmuka *repository* tetap konsisten. Dengan demikian, penggunaan *pattern repository abstraction* berhasil memungkinkan transisi yang mulus dari penyimpanan *in-memory* ke penyimpanan database. Operasi-operasi seperti penambahan stok, pengurangan stok, pembuatan reservasi, pembatalan reservasi, tetap berjalan sesuai desain domain, tetapi kini perubahan data dicatat secara permanen dalam *database*, bukan hanya bertahan selama aplikasi berjalan. Integrasi ini juga membuka peluang pengembangan lebih lanjut, seperti integrasi ke modul *Reporting & Analytics* atau implementasi *event sourcing* pada *deliverable* selanjutnya.

# IMPLEMENTASI DOMAIN MODEL

Implementasi domain model bertujuan untuk merepresentasikan aturan bisnis inti dari pengelolaan inventori sesuai prinsip Domain-Driven Design (DDD). Domain model ditempatkan di dalam folder `src/domain/` dan terdiri dari komponen-komponen utama seperti Aggregate Root, Value Object, serta Entity, yang bekerja bersama untuk memastikan konsistensi data dan perilaku sistem.

## A. Aggregate Root: `InventoryItem`

Aggregate utama pada modul ini adalah `InventoryItem`, yang berfungsi sebagai pusat logika bisnis terkait pengelolaan stok barang. Aggregate ini menyimpan informasi mengenai jumlah stok tersedia (`on_hand`), stok yang sedang dipesan (`reserved`), ambang batas stok minimum (`threshold`), serta catatan pergerakan stok (`moves`). Beberapa operasi penting yang diimplementasikan pada aggregate ini meliputi:

- `increase()` – menambah jumlah stok barang berdasarkan `Quantity`.
- `decrease()` – mengurangi stok yang tersedia, dengan validasi agar tidak melebihi stok `available`.
- `reserve()` – menyimpan stok untuk pesanan tertentu dan menambah daftar `reservation`.
- `release()` – membatalkan `reservation` tertentu dan mengembalikannya ke stok `available`.
- `adjust()` – menyesuaikan stok secara manual (positif atau negatif).

Setiap perubahan menerapkan *invariant checks* untuk memastikan bahwa nilai `on_hand`, `reserved`, dan `available` tetap valid dan konsisten.

## B. Value Objects

Value Objects digunakan untuk merepresentasikan nilai-nilai yang tidak memiliki identitas unik dan ditentukan oleh nilainya.

- SKU - Digunakan untuk merepresentasikan kode produk. Dibuat immutable dan divalidasi agar tidak kosong.
- Quantity - Mewakili jumlah stok beserta satuannya (`uom`). Mendukung operasi penjumlahan (`add`) dan pengurangan (`sub`) dengan validasi UOM dan mencegah hasil negatif.
- Threshold - Digunakan untuk menentukan batas minimal stok sebelum dianggap low stock.
- Batch - Menyimpan informasi batch produksi dan tanggal kedaluwarsa jika diperlukan di masa depan.

Penggunaan value object ini memastikan konsistensi data dan mencegah duplikasi logika validasi.

## C. Entities

Entitas digunakan untuk objek yang memiliki identitas unik selama daur hidupnya. Identitas ini tidak berubah meskipun atribut lain pada entitas tersebut mengalami perubahan seiring berjalannya proses bisnis. Dengan kata lain, entitas diperlakukan sebagai objek yang tetap sama sepanjang waktu, bukan berdasarkan nilai-nilai atributnya, tetapi berdasarkan identity, biasanya berupa ID unik.

- Reservation - Digunakan untuk menyimpan stok yang dialokasikan untuk sebuah pesanan (`order_id`). Setiap reservation memiliki ID unik dan jumlah barang yang dicadangkan.

- StockMove - Mewakili pergerakan stok, baik itu masuk (IN), keluar (OUT), maupun penyesuaian (ADJUST). Informasi seperti jumlah, tipe pergerakan, alasan, dan timestamp dicatat sebagai histori.

Kedua entitas ini membantu menjaga histori dan memastikan integritas operasi reserve, release, increase, dan decrease. Dengan diterapkannya domain model yang kuat, sistem dapat menjamin bahwa setiap operasi yang terjadi pada inventori mengikuti aturan bisnis yang benar dan terhindar dari inkonsistensi data. Model ini juga menjadi pondasi bagi Application Layer dan API Layer sehingga seluruh alur kerja Inventory Control dapat berjalan secara konsisten dan terstruktur.

Meskipun *deliverable 5* memperkenalkan fitur autentikasi berbasis JWT serta migrasi *repository* menuju *database*, struktur domain model pada modul *Inventory Control* tidak mengalami perubahan. Seluruh komponen inti, termasuk *aggregate* *InventoryItem*, *entity* seperti *Reservation* dan *StockMove*, serta *value object* seperti *SKU*, *Quantity*, *Threshold*, dan *Batch*, tetap dipertahankan sebagaimana dirancang pada *deliverable 3* dan diimplementasikan pada *deliverable 4*. Hal ini dilakukan untuk menjaga konsistensi aturan bisnis dan memastikan bahwa perluasan sistem pada tahap ini hanya menyentuh lapisan infrastruktur dan API, tanpa mempengaruhi logika domain yang sudah stabil dan tervalidasi.

## IMPLEMENTASI APPLICATION LAYER

Application Layer berfungsi sebagai penghubung antara domain model dan antarmuka API. Lapisan ini mengatur alur eksekusi use case, mengoordinasikan pemanggilan metode pada aggregate, serta menangani penyimpanan data melalui repository. Dalam implementasi modul Inventory Control, Application Layer ditempatkan pada folder `src/services/` dan terdiri dari dua komponen utama, yaitu repository dan service.

### A. Repository: `InventoryRepositoryInMemory`

Repository berfungsi sebagai abstraksi mekanisme penyimpanan data sehingga domain maupun API tidak bergantung pada detail teknis seperti database atau file. Pada tahap implementasi awal ini, digunakan repository berbasis in-memory, yang menyimpan data menggunakan struktur Python dictionary. Walaupun sederhana, pendekatan ini efektif untuk memastikan fungsi domain dapat dieksekusi dan diuji tanpa ketergantungan sistem eksternal. Repository menyediakan beberapa operasi dasar:

**Tabel 4.** Method pada `InventoryRepositoryInMemory`

Method	Fungsi
<code>list_all()</code>	Mengembalikan semua item inventori yang tersimpan
<code>get_by_id()</code>	Mengambil item berdasarkan ID
<code>get_by_sku()</code>	Mengambil item berdasarkan SKU
<code>save()</code>	Menyimpan atau memperbarui item dalam repository

Dengan memisahkan penyimpanan lewat repository, implementasi ini dapat dengan mudah dikembangkan ke database sesungguhnya pada milestone berikutnya tanpa mengubah logika domain. Strategi ini juga sejalan dengan peran *Inventory Control* sebagai *Open Host Service* pada *context map*, yang menuntut sumber data stok yang stabil dan konsisten untuk dikonsumsi konteks lain. Dengan menjaga fleksibilitas ini, implementasi pada *deliverable* berikutnya dapat memperkenalkan *database* yang sesungguhnya dengan tetap mempertahankan struktur domain dan arsitektur aplikasi yang telah dirancang.

### B. Service: `InventoryService`

`InventoryService` merupakan komponen inti pada Application Layer yang bertanggung jawab menjalankan use case pengelolaan stok. Service ini memanfaatkan repository untuk mengambil atau menyimpan data, kemudian memanggil metode pada aggregate `InventoryItem` untuk menerapkan aturan bisnis yang sesuai. Service menyediakan berbagai operasi penting, di antaranya:

**Tabel 5.** Use Case Application Layer

Kategori Use Case	Nama Use Case	Deskripsi Fungsi
<b>Admin Gudang</b>	<code>create_item</code>	Membuat item baru dengan stok awal, satuan (uom), dan batas minimum stok.

	<code>list_items</code>	Menampilkan seluruh item inventori yang tersedia di sistem.
	<code>get_item</code>	Mengambil informasi lengkap item berdasarkan SKU.
	<code>set_threshold</code>	Mengubah nilai minimum stok (threshold) untuk suatu item.
<b>Inbound / Outbound</b>	<code>increase_stock</code>	Menambah stok ketika barang masuk (goods received / inbound).
	<code>decrease_stock</code>	Mengurangi stok ketika barang keluar (outbound / shipment).
	<code>adjust_stock</code>	Melakukan koreksi stok manual, baik penambahan maupun pengurangan (stock opname).
<b>Order / Reservation</b>	<code>reserve_stock</code>	Melakukan reservasi stok untuk suatu order sehingga stok disisihkan.
	<code>release_reservation</code>	Membatalkan reservasi dan mengembalikan stok ke available.
<b>Monitoring</b>	<code>get_availability</code>	Memberikan informasi ketersediaan stok: <code>on_hand</code> , <code>reserved</code> , dan <code>available</code> .
	<code>get_low_stock_items</code>	Menampilkan daftar item yang berada di bawah batas minimum stok (low stock).

Setiap use case ini mengatur interaksi antara API layer dan domain model, memastikan bahwa aturan bisnis dalam domain dijalankan secara konsisten dan semua perubahan inventori tersimpan dengan benar melalui repository. Dengan struktur Application Layer yang terpisah ini, kode menjadi lebih modular, mudah diuji, dan mendukung prinsip separation of concerns. Lapisan ini juga memastikan bahwa domain tetap bersih dari detail teknis, dan API dapat berinteraksi dengan domain melalui antarmuka yang jelas dan terstruktur.

Pada *deliverable 5*, struktur *Application Layer* tetap dipertahankan sebagaimana pada implementasi sebelumnya, namun mekanisme penyimpanan data kini dialihkan dari `InventoryRepositoryInMemory` ke *repository* berbasis database (`InventoryRepositoryDB`). Perubahan ini tidak memengaruhi logika use case maupun interaksi antara *service* dan domain model, karena antarmuka *repository* tetap konsisten sesuai rancangan awal. Dengan demikian, seluruh operasi seperti penambahan stok, pengurangan stok, reservasi, dan *monitoring* tetap berjalan melalui `InventoryService` tanpa perubahan perilaku, sementara penyimpanan data kini dilakukan secara persisten melalui SQLAlchemy. Integrasi ini memperkuat Application Layer tanpa mengubah struktur inti, sekaligus membuktikan bahwa desain repository abstraction pada *deliverable 4* berhasil mendukung migrasi penyimpanan pada tahap implementasi lanjutan.

## IMPLEMENTASI API LAYER

API Layer berfungsi sebagai antarmuka antara sistem Inventory Control dengan dunia luar, baik pengguna manusia (misalnya admin gudang melalui tool seperti Postman) maupun sistem lain (seperti modul Order, Inbound, atau Outbound). Pada implementasi ini, API dibangun menggunakan framework FastAPI dan didefinisikan di dalam file `src/main.py`. Lapisan ini memetakan setiap endpoint HTTP ke use case yang telah disediakan oleh `InventoryService`.

### A. Struktur Umum API

Aplikasi FastAPI diinisialisasi dengan membuat instance FastAPI dan kemudian mendefinisikan beberapa kelompok endpoint berdasarkan aktor atau konteks pengguna:

- Endpoint Admin → prefix `/admin/...`  
Digunakan oleh admin atau staf gudang untuk mengelola master data item dan pengaturan stok.
- Endpoint OHS (Open Host Service) → prefix `/ohs/...`  
Digunakan oleh sistem lain (Order, Inbound, Outbound) untuk berinteraksi dengan Inventory Control.
- Endpoint Manager → prefix `/manager/...`  
Digunakan oleh manajer untuk memantau kondisi stok dan low stock.

Setiap endpoint menggunakan model Pydantic dari `src/schemas/inventory.py` sebagai request body dan response DTO, sehingga data yang masuk dan keluar API tervalidasi secara otomatis.

### B. Endpoint Admin Gudang (`/admin/...`)

Kelompok endpoint ini memanfaatkan use case kategori Admin Gudang dari `InventoryService`, seperti `create_item`, `list_items`, `get_item`, `set_threshold`, dan `adjust_stock`.

**Tabel 6. Endpoint Admin Gudang**

Endpoint	Method	Use Case	Deskripsi
<code>/admin/items</code>	POST	<code>create_item</code>	Membuat item baru dengan stok awal dan threshold.
<code>/admin/items</code>	GET	<code>list_items</code>	Mengambil seluruh item inventori.
<code>/admin/items/{sku}</code>	GET	<code>get_item</code>	Mengambil detail satu item berdasarkan SKU.
<code>/admin/items/{sku}/threshold</code>	POST	<code>set_threshold</code>	Mengubah batas minimum stok (threshold) suatu item.
<code>/admin/items/{sku}/adjust</code>	POST	<code>adjust_stock</code>	Melakukan penyesuaian stok manual (stock opname).

Di dalam implementasi, endpoint menerima request dalam bentuk `CreateItemRequest`, `SetThresholdRequest`, atau `AdjustStockRequest`, lalu memanggil *method service* yang sesuai dan mengembalikan response dalam bentuk `InventoryItemDto`.

## C. Endpoint Open Host Service (/ohs/...)

Kelompok endpoint ini merepresentasikan API yang seolah-olah dipakai oleh *bounded context* lain untuk berinteraksi dengan *Inventory Control*. Fungsinya lebih ke operasional harian (*inbound*, *outbound*, reservasi).

**Tabel 7. Endpoint OHS**

Endpoint	Method	Use Case	Deskripsi
/ohs/{sku}/increase	POST	increase_stock	Menambah stok saat barang masuk (inbound).
/ohs/{sku}/decrease	POST	decrease_stock	Mengurangi stok saat barang keluar (outbound).
/ohs/{sku}/reserve	POST	reserve_stock	Melakukan reservasi stok untuk order tertentu.
/ohs/{sku}/release	POST	release_reservation	Membatalkan reservasi tertentu berdasarkan reservation ID.
/ohs/availability/{sku}	GET	get_availability	Mengambil informasi on_hand, reserved, dan available untuk sebuah SKU.
/ohs/{sku}/reservations	GET	(akses ke item.reservations)	Menampilkan daftar reservasi untuk suatu SKU.

Setiap endpoint terhubung langsung dengan use case pada *InventoryService*:

- *increase\_stock* digunakan dalam alur inbound (*goods received*),
- *decrease\_stock* digunakan pada proses *outbound* atau pengeluaran barang,
- *reserve\_stock* digunakan oleh Order untuk mengalokasikan stok,
- *release\_reservation* digunakan untuk membatalkan alokasi stok,
- *get\_availability* dan *endpoint* reservasi digunakan untuk kebutuhan pengecekan ketersediaan stok secara *real-time*.

Body request untuk endpoint ini menggunakan schema seperti *IncreaseStockRequest*, *DecreaseStockRequest*, *ReserveStockRequest*, dan *ReleaseReservationRequest*. API kemudian mengembalikan data baik dalam bentuk *InventoryItemDto* atau *InventoryStats*, tergantung kebutuhan.

## D. Endpoint Manager (/manager/...)

Kelompok endpoint ini menyediakan informasi untuk kebutuhan monitoring level manajerial. Endpoint pada kelompok ini mengacu pada *use case monitoring* dari *InventoryService*, yaitu *get\_low\_stock\_items*.

**Tabel 8. Endpoint Manager**

Endpoint	Method	Use Case	Deskripsi
----------	--------	----------	-----------

/manager/low-stock	GET	get_low_stock_items	Mengembalikan daftar item yang berada di bawah threshold (low stock).
--------------------	-----	---------------------	-----------------------------------------------------------------------

*Endpoint* ini mengembalikan list `InventoryItemDto` sehingga manajer dapat melihat `SKU`, `stok on hand`, `reserved`, `available`, serta informasi `threshold` dan status `low_stock`.

## E. Integrasi dengan Pydantic Schema

Setiap endpoint menggunakan Pydantic model dari `schemas/inventory.py` untuk memisahkan representasi domain dengan bentuk data yang ditransfer melalui API. Contohnya:

- `CreateItemRequest` untuk input pembuatan item.
- `IncreaseStockRequest`, `DecreaseStockRequest`, `AdjustStockRequest` untuk operasi stok.
- `InventoryItemDto` dan `InventoryStats` untuk response yang dikembalikan ke klien.

Pemakaian DTO ini memastikan bahwa domain model (`InventoryItem`, `Reservation`, dll.) tidak terekspos langsung ke dunia luar, sekaligus menjaga konsistensi format data untuk dokumentasi otomatis di Swagger UI.

## F. Integrasi Autentikasi & Role-Based Access Control

Pada *deliverable 5*, API *Layer* mengalami perluasan fungsi melalui penerapan autentikasi berbasis JSON Web Token (JWT) dan mekanisme otorisasi berbasis peran (*role-based access control*). Jika pada *deliverable 4* seluruh endpoint dapat diakses secara bebas, maka pada tahap pengembangan ini setiap permintaan harus disertai token yang valid, dan akses *endpoint* dibatasi berdasarkan peran pengguna, yaitu admin, manager, dan client. Perubahan ini tidak mengubah struktur endpoint maupun alur *use case* yang sudah ditetapkan pada *deliverable 4*, tetapi menambahkan lapisan keamanan yang memastikan bahwa operasi tertentu hanya dapat dilakukan oleh pihak yang berwenang.

Integrasi autentikasi diterapkan melalui dependency `get_current_user`, yang bertugas memverifikasi token, mengecek masa berlaku, dan memastikan token tidak berada dalam daftar *blacklist*. Setelah token tervalidasi, pemeriksaan lanjutan dilakukan melalui fungsi `require_role(...)` yang memastikan peran pengguna sesuai dengan *endpoint* yang diakses. Dengan mekanisme ini, endpoint dengan prefix `/admin/...` hanya dapat dipanggil oleh pengguna yang terdaftar sebagai admin, endpoint `/manager/...` hanya dapat diakses oleh peran manager, dan endpoint `/ohs/...` dibatasi untuk peran *client* yang mewakili sistem eksternal seperti *Inbound*, *Outbound*, atau *Order Management*.

Selain itu, API *Layer* kini terhubung dengan *Application Layer* yang menggunakan database sebagai backend penyimpanan, sehingga setiap respons API mencerminkan kondisi data yang tersimpan secara persisten. Walaupun fungsi endpoint tetap sama seperti *deliverable 4*, integrasi *repository database* membuat perubahan pada stok, reservasi, atau threshold kini tercatat dalam database SQLite melalui SQLAlchemy. Dengan demikian, API *Layer* tidak hanya menjalankan peran sebagai penghubung ke logika domain, tetapi juga berfungsi sebagai lapisan keamanan dan akses kontrol yang menjaga integritas dan keamanan data modul *Inventory Control*.

## **UNIT TESTING DAN TEST DRIVEN DEVELOPMENT (TDD)**

*Unit Test* berfungsi sebagai mekanisme untuk memastikan setiap komponen atau modul dari sistem *Inventory Control* bekerja sesuai spesifikasi, baik secara terpisah maupun dalam integrasi minimal. Pendekatan ini memungkinkan pengembang untuk memverifikasi logika bisnis di *InventoryService*, validasi *payload*, serta interaksi dengan *repository* dan *database* tanpa harus menjalankan seluruh aplikasi. Pada implementasi ini, unit test ditulis menggunakan *framework pytest* dan diletakkan di dalam folder `tests/`. Setiap *test case* memetakan skenario input tertentu ke hasil yang diharapkan, mendukung pendekatan *Test-Driven Development* (TDD), di mana pengujian ditulis sebelum implementasi fitur untuk memastikan kode yang dikembangkan memenuhi kebutuhan fungsional dan menangani kasus error dengan benar.

### **A. Konsep TDD dan Test Coverage**

Pendekatan *Test Driven Development* (TDD) diterapkan untuk memastikan bahwa setiap fitur yang dikembangkan memenuhi spesifikasi dan dapat berjalan dengan benar sejak awal. TDD mendorong pengembang untuk menulis *unit test* terlebih dahulu sebelum implementasi kode. Dengan cara ini, pengujian menjadi bagian integral dari proses pengembangan, bukan sekadar langkah tambahan setelah kode selesai dibuat. Alur TDD pada layanan *inventory control* dapat dijelaskan sebagai berikut:

#### **1) Menulis *Test Case* Terlebih Dahulu**

Setiap fitur atau metode yang akan dikembangkan, seperti validasi SKU, penambahan atau pengurangan stok, serta mekanisme reservasi item, diuji terlebih dahulu melalui *test case* yang merepresentasikan perilaku yang diharapkan.

#### **2) Menjalankan *Test Case***

Pada tahap awal, *test case* akan gagal (*fail*) karena implementasi kode untuk fitur tersebut belum tersedia. Kegagalan ini menegaskan bahwa *test case* benar-benar memvalidasi kondisi yang diinginkan.

#### **3) Mengimplementasikan Kode**

Pengembang kemudian menulis kode yang diperlukan agar *test case* dapat dilewati. Implementasi dilakukan dengan fokus pada memenuhi semua persyaratan fungsional dan menjaga konsistensi domain model.

#### **4) Menjalankan *Test Case* Kembali**

Setelah implementasi, test dijalankan ulang. Jika test berhasil (*pass*), berarti fitur berfungsi sesuai yang diharapkan. Proses ini memastikan bahwa setiap perubahan kode selalu terverifikasi.

#### **5) Iterasi Berkelanjutan**

Langkah-langkah di atas diulang untuk setiap fitur baru. Dengan demikian, setiap bagian dari sistem diuji secara sistematis dan berkesinambungan.

Dengan penerapan TDD, setiap perubahan kode selalu melalui pengujian, sehingga potensi kesalahan dapat terdeteksi lebih awal dan perbaikan dapat dilakukan sebelum masalah berkembang lebih jauh. Selain itu, TDD memungkinkan pengukuran *test coverage*, yaitu persentase kode yang tercakup oleh *unit test*. *Test coverage* ini mencakup berbagai kondisi, termasuk kasus normal, *edge cases*, penanganan error, *workflow*, dan *invariant checks*, sehingga menjamin kualitas dan stabilitas sistem secara menyeluruh.

### **B. Struktur Unit Test**

Struktur folder dan file *unit test* pada proyek ini dirancang agar modular, terorganisir, dan mudah dikembangkan. Struktur saat ini adalah sebagai berikut:

```
tests/
└── conftest.py
└── test_domain_inventory.py
└── test_inventory_service.py
```

### 1. `conftest.py`

File ini disiapkan untuk mendefinisikan *fixture* global yang dapat digunakan oleh semua *file test*. Meskipun saat ini `conftest.py` masih kosong, `pytest` tetap dapat mengeksekusi *test* karena fixture juga bisa langsung didefinisikan di masing-masing *file test*.

### 2. `test_domain_inventory.py`

Berisi unit test untuk domain model, termasuk entitas seperti `SKU`, `Quantity`, dan `InventoryItem`. Test pada file ini fokus pada validasi aturan domain, pengelolaan stok, dan invariant checks.

### 3. `test_inventory_service.py`

Berisi *unit test* untuk *application/service layer*, khususnya `InventoryService`. Test ini memverifikasi alur bisnis, interaksi dengan *repository*, serta mekanisme penambahan, pengurangan, dan reservasi stok.

Penggunaan `conftest.py`, meskipun kosong saat ini, memberikan fleksibilitas jangka panjang. Di masa depan, jika diperlukan *fixture* atau konfigurasi yang digunakan oleh banyak *test*, cukup ditambahkan di `conftest.py` tanpa harus mengubah setiap *file test*. Hal ini menjaga modularitas, memudahkan pemeliharaan, dan memastikan bahwa struktur *test* tetap *scalable* seiring pertumbuhan proyek.

## C. Contoh *Test Cases*

*Test cases* yang dikembangkan untuk proyek ini dirancang agar menyeluruh, konsisten, dan mencakup seluruh alur bisnis, termasuk kasus normal, batas (*edge cases*), dan skenario *error*. Secara garis besar, cakupannya dibagi menjadi beberapa kategori:

### 1. Domain Model

*Unit test* pada level domain memastikan bahwa aturan domain dipatuhi dan entitas berperilaku sesuai spesifikasi:

- `SKU`: validasi agar tidak kosong atau hanya berisi *whitespace*.
- `Quantity`: nilai tidak boleh negatif; operasi penjumlahan (*add*) dan pengurangan (*sub*) harus konsisten dengan *unit of measure*.
- `InventoryItem`: memeriksa perhitungan *available stock*, operasi *increase/decrease stock*, penyesuaian stok melalui *adjust*, pengecekan *low stock*, pengelolaan reservasi, dan *invariant checks* untuk memastikan integritas data.

### 2. Service Layer

*Test* pada lapisan *service* memverifikasi bahwa logika bisnis dieksekusi dengan benar dan repository berinteraksi sesuai kontrak. `InventoryService` harus mampu melakukan pembuatan item baru, pencegahan duplikasi `SKU`, penambahan/pengurangan stok, reservasi dan pelepasan stok, penyesuaian stok (*adjust*), pengaturan *threshold*, serta pengambilan item yang stoknya rendah atau ketersediaannya.

### 3. Edge Cases & Error Handling

Test juga dirancang untuk menangkap kondisi ekstrem dan kesalahan:

- *Adjustment* negatif yang melebihi stok yang tersedia.
- *Reserved stock* yang melebihi *on-hand stock*.
- Ketidaksesuaian unit of measure (*UOM mismatch*).
- Percobaan pelepasan reservasi yang tidak ada.

Dengan cakupan test ini, seluruh alur domain dan *service* teruji secara menyeluruh, termasuk kondisi normal, batas, dan skenario error. Pendekatan ini memastikan integritas sistem, mendukung *Test Driven Development* (TDD), dan memberikan kepastian bahwa perubahan kode tidak merusak fungsi yang sudah ada.

#### **D. Hasil *Coverage***

Setelah semua unit test dibuat dan dijalankan, penting untuk mengevaluasi *test coverage*, yaitu persentase kode yang tercakup oleh *unit test*. *Test coverage* yang tinggi menunjukkan bahwa hampir seluruh alur, kondisi, dan skenario error telah diuji, sehingga meningkatkan keandalan dan keamanan aplikasi.

##### 1. Menjalankan *Coverage*

Untuk menghitung *coverage*, digunakan perintah berikut pada terminal:

```
pytest -vv --color=yes --cov=src --cov-report=term-missing

# -vv : menampilkan output pytest dengan tingkat verbose tinggi, sehingga
setiap test case ditampilkan.
# --color=yes : menambahkan warna pada output untuk membedakan test yang
pass dan fail.
# --cov=src : menghitung coverage pada direktori src (kode utama aplikasi).
# --cov-report=term-missing : menampilkan laporan coverage di terminal dan
menandai baris kode yang belum diuji.
```

##### 2. Interpretasi Hasil

*Output command* akan menunjukkan:

- Jumlah *test* yang dijalankan, lulus, atau gagal.
- Persentase kode yang tercakup (misal: 100%).
- Baris kode yang belum tercover (jika ada).

```

Ivana@APTOP-0GGCBACU MINGW64 /d/113160_TB_18223126_WMS (main)
$ pytest -vv --color=yes --cov=src --cov-report=term-missing
=====
test session starts =====
platform win32 -- Python 3.13.5, pluggy 1.6.0 -- C:\Users\Ivana\AppData\Local\Programs\Python\Python313\python.exe
cachedir: .pytest_cache
rootdir: D:\113160_TB_18223126_WMS
plugins: anyio-4.11.6, asyncio-0.23.5, cov-4.1.0
asyncio: mode=Mode.STRICT
collected 54 items

src/tests/test_domain_inventory.py::test_sku_not_empty PASSED
src/tests/test_domain_inventory.py::test_quantity_negative PASSED
src/tests/test_domain_inventory.py::test_quantity_add PASSED
src/tests/test_domain_inventory.py::test_quantity_sub PASSED
src/tests/test_domain_inventory.py::test_quantity_sub_negative PASSED
src/tests/test_domain_inventory.py::test_inventory_available PASSED
src/tests/test_domain_inventory.py::test_inventory_increase PASSED
src/tests/test_domain_inventory.py::test_inventory_decrease PASSED
src/tests/test_domain_inventory.py::test_inventory_decrease_not_enough PASSED
src/tests/test_domain_inventory.py::test_low_stock PASSED
src/tests/test_domain_inventory.py::test_inventory_adjust_positive PASSED
src/tests/test_domain_inventory.py::test_inventory_adjust_negative PASSED
src/tests/test_domain_inventory.py::test_inventory_adjust_zero PASSED
src/tests/test_domain_inventory.py::test_inventory_adjust_negative_beyond_stock PASSED
src/tests/test_domain_inventory.py::test_inventory_adjust_keeps_invariants PASSED
src/tests/test_domain_inventory.py::test_inventory_invariants_reserved_greater_than_onhand PASSED
src/tests/test_domain_inventory.py::test_sku_strip_invalid PASSED
src/tests/test_domain_inventory.py::test_inventory_adjust_zero_does_not_change_stock PASSED
src/tests/test_domain_inventory.py::test_inventory_adjust_large_positive PASSED
src/tests/test_domain_inventory.py::test_sku_strip_and_normalize PASSED
src/tests/test_domain_inventory.py::test_inventory_release_success PASSED
src/tests/test_domain_inventory.py::test_inventory_adjust_negative_overflow PASSED
src/tests/test_domain_inventory.py::test_release_reservation_success PASSED
src/tests/test_domain_inventory.py::test_quantity_add uom_mismatch PASSED
src/tests/test_domain_inventory.py::test_quantity_sub uom_mismatch PASSED
src/tests/test_domain_inventory.py::test_invariant_on_hand_negative PASSED
src/tests/test_domain_inventory.py::test_invariant_reserved_negative PASSED
src/tests/test_inventory_service.py::test_create_item PASSED
src/tests/test_inventory_service.py::test_create_duplicate_sku PASSED
src/tests/test_inventory_service.py::test_increase_stock PASSED
src/tests/test_inventory_service.py::test_decrease_stock PASSED
src/tests/test_inventory_service.py::test_decrease_stock_fail PASSED
src/tests/test_inventory_service.py::test_reservation PASSED
src/tests/test_inventory_service.py::test_reservation_not_found PASSED
src/tests/test_inventory_service.py::test_adjust_negative_delta PASSED
src/tests/test_inventory_service.py::test_invariant_reserved_exceeds_onhand PASSED
src/tests/test_inventory_service.py::test_is_low_stock_exact_border PASSED
src/tests/test_inventory_service.py::test_get_item_not_found PASSED
src/tests/test_inventory_service.py::test_set_threshold PASSED
src/tests/test_inventory_service.py::test_adjust_stock PASSED
src/tests/test_inventory_service.py::test_get_availability PASSED
src/tests/test_inventory_service.py::test_get_low_stock_items PASSED

----- warnings summary -----
src/tests/test_inventory_service.py: 10 warnings
src/tests/test_inventory_service.py: 7 warnings
<string>:6: DeprecationWarning: datetime.datetime.utcnow() is deprecated and scheduled for removal in a future version. Use timezone-aware objects to represent datetimes in UTC: datetime.datetime.now(datetime.UTC).
-- Docs: https://docs.pytest.org/en/stable/how-to/capture-warnings.html
----- coverage: platform win32, python 3.13.5-final-0 -----
Name           Stmts  Miss Cover  Missing
src\domain\inventory.py      117    0   100%
src\services\inventory_service.py  47    0   100%
TOTAL          164    0   100%
===== 54 passed, 17 warnings in 0.36s =====

```

Gambar 1. Hasil *Test Coverage*

Dengan cakupan 100%, dapat dipastikan bahwa semua metode utama, validasi domain, operasi stok, dan skenario error telah diuji. Bahkan jika terdapat file `conftest.py` yang kosong, `pytest` tetap dapat menjalankan semua test karena *fixture* digunakan langsung di masing-masing *file test*. Hal ini memungkinkan pengembangan *test* yang modular dan fleksibel, serta memudahkan penambahan *fixture* global di masa depan tanpa menurunkan *coverage*.

### 3. Manfaat *Coverage* dalam TDD

- Deteksi awal bug → setiap perubahan kode dijamin diuji secara otomatis.
- Keamanan dan stabilitas → perubahan baru tidak merusak fungsionalitas yang sudah ada.
- Dokumentasi perilaku sistem → test cases berfungsi sebagai referensi perilaku sistem untuk developer lain.
- Peningkatan kualitas kode → mengidentifikasi bagian kode yang kurang diuji atau kompleksitas berlebihan.

Dengan pendekatan ini, integrasi TDD, unit test, dan *coverage* memastikan pengembangan *service inventory control* lebih aman, dapat diprediksi, dan berkualitas tinggi.

# CI/CD DENGAN GITHUB WORKFLOWS

Implementasi *Continuous Integration (CI)* dan *Continuous Deployment (CD)* pada *project inventory control* dilakukan menggunakan GitHub Workflows. Pendekatan ini memungkinkan setiap perubahan kode diuji, diperiksa kualitasnya, dibangun, dan dikemas secara otomatis sebelum di-*merge* atau di *deploy*.

## A. Struktur Workflow

Struktur *folder workflow* pada *repository* adalah sebagai berikut:

```
.github/  
└── workflows/  
    └── ci.yml
```

File *ci.yml* mendefinisikan *pipeline* CI/CD, termasuk langkah-langkah *linting*, *testing*, dan *build Docker* (jika menggunakan *container*). Setiap *commit* atau *pull request* ke *branch* utama akan memicu *workflow* ini secara otomatis.

## B. Tahapan Workflow

*Workflow* terdiri dari beberapa tahap utama:

### 1. Linting

- *Linter* adalah proses otomatis untuk memeriksa konsistensi kode, gaya penulisan, dan potensi *error* sebelum kode dijalankan atau di-*merge*. Tujuan *linting* adalah menjaga agar kode rapi, mudah dibaca, dan bebas dari kesalahan sintaks atau praktik pemrograman yang buruk.
- Beberapa tools yang digunakan dalam project ini:
  - a. Flake8: Memeriksa kesalahan sintaks, bug potensial, dan pelanggaran PEP 8 (Python Enhancement Proposal 8, panduan gaya Python). Contoh masalah yang dideteksi: variabel tidak terpakai, import yang tidak digunakan, indentasi tidak konsisten.
  - b. Black: Code formatter otomatis yang memformat kode agar konsisten. Black memastikan penulisan kode memiliki indentasi, spasi, dan pemisahan baris yang seragam di seluruh project, sehingga memudahkan kolaborasi tim.
  - c. Isort: Mengurutkan import secara otomatis agar lebih rapi dan mudah dibaca. Memisahkan import standar Python, import pihak ketiga, dan import lokal sesuai urutan yang konsisten.
- Dengan kombinasi ketiga tools ini, setiap commit yang masuk ke *repository* dijamin bebas dari error sintaks sederhana, mematuhi standar gaya Python, memiliki struktur import yang rapi.
- Dalam GitHub Workflow, tahap linting dijalankan sebelum *unit test* dan *build Docker*, sehingga *developer* dapat segera memperbaiki isu gaya kode atau potensi error sebelum menguji atau mendeploy kode.

### 2. Unit Testing dan Coverage

- Menjalankan *test* dengan *pytest* dan menghitung *coverage*:

```
pytest -vv --color=yes --cov=src --cov-report=term-missing
```

- Menjamin bahwa setiap perubahan kode tetap memenuhi 100% *test coverage*, termasuk *edge cases*, *error handling*, *workflow*, dan *invariant checks*.

### 3. Build dan Run Docker

- *Dockerfile* disiapkan untuk membangun image dari *service inventory control*.
- Container ini memuat seluruh *dependency*, database, dan *service* agar dapat dijalankan secara konsisten di berbagai lingkungan.

- *Workflow* memastikan Docker image berhasil dibuat dan *service* berjalan di container:

```
docker build -t inventory-service .
docker run --rm -p 8000:8000 inventory-service
```

- Implementasi Docker menjamin isolasi lingkungan sehingga *test* yang dijalankan di CI menyerupai lingkungan produksi, meminimalkan risiko bug akibat perbedaan *environment*.

```
Ivana@LAPTOP-06GCRACU MINGW64 /d/II3160_TB_18223126_WMS (main)
● $ docker build -t inventory-service .
[+] Building 5.3s (10/10) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 400B
=> [internal] load metadata for docker.io/library/python:3.11-slim
=> [internal] load .dockerrcignore
=> => transferring context: 28B
=> [1/5] FROM docker.io/library/python:3.11-slim@sha256:7cd0079a9bd8800c81632d65251048fc2848bf9afda542224b1b10e0cae45575
=> => resolve docker.io/library/python:3.11-slim@sha256:7cd0079a9bd8800c81632d65251048fc2848bf9afda542224b1b10e0cae45575
=> [internal] load build context
=> => transferring context: 120.10kB
=> CACHED [2/5] WORKDIR /app
=> CACHED [3/5] COPY requirements.txt .
=> CACHED [4/5] RUN pip install --no-cache-dir -r requirements.txt
=> [5/5] COPY .
=> exporting to image
=> => exporting layers
=> => exporting manifest sha256:9f675026c085aedb02090ca2aa8b90fe911bbddf30b413004827f1573fac22ff
=> => exporting config sha256:018ab60f96f58490bfdebf5f46d59f1ac2c9923c68ea07aa6b132d0acb8c1d0
=> => exporting attestation manifest sha256:9a58019e6148c8dd961fb45fabf1b4a08a96a912f71a8839c389486dc0d6a1
=> => exporting manifest list sha256:e9d45ec59f9c9c3d4a45b57fe44ac0614818e8900b6f8ef0dabd8cb34b141cd
=> => naming to docker.io/library/inventory-service:latest
=> => unpacking to docker.io/library/inventory-service:latest

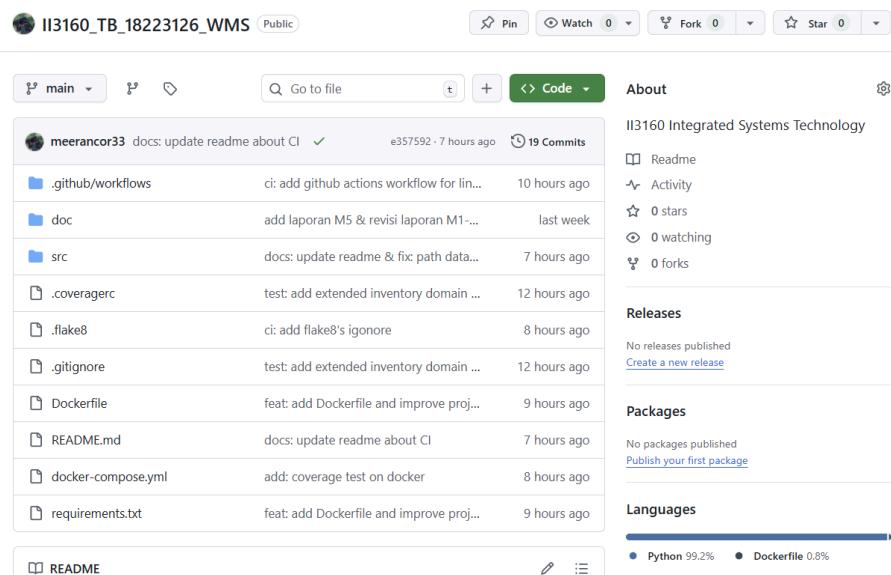
Ivana@LAPTOP-06GCRACU MINGW64 /d/II3160_TB_18223126_WMS (main)
● $ docker run --rm -p 8000:8000 inventory-service
INFO: Started server process [1]
INFO: Waiting for application startup.
WARNING:root:
=====
FastAPI server is running inside Docker
Open: http://localhost:8000/docs
Health check: http://localhost:8000/health
Note: http://0.0.0.0:8000 TIDAK bisa dibuka dari browser
=====

INFO: Application startup complete.
INFO: Unicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
INFO: 172.17.0.1:51058 - "GET /health HTTP/1.1" 200 OK
INFO: 172.17.0.1:51058 - "GET /docs HTTP/1.1" 200 OK
INFO: 172.17.0.1:51058 - "GET /openapi.json HTTP/1.1" 200 OK
INFO: 172.17.0.1:51072 - "GET /health HTTP/1.1" 200 OK
```

Gambar 2. Hasil Ketika FastAPI Berhasil Berjalan dalam Docker

#### 4. Deployment atau *Checklist* Indikator

- *Workflow* dapat menandai branch/repo dengan *checklist* “Service Ready” setelah semua *test* lulus dan build sukses.
- *Checklist* ini menjadi indikator bahwa *service* siap dijalankan di lingkungan produksi atau *staging*.
- Selain di repo, indikator keberhasilan CI juga dapat dilihat secara langsung di tab “Actions” pada GitHub. Setiap *workflow* yang dijalankan akan menampilkan status (*success*, *failed*, *in progress*) beserta *log* dari setiap langkah, sehingga *developer* bisa memantau proses *linting*, *testing*, dan *build* secara *real-time*.



All workflows	Event	Status	Branch	Actor
Showing runs from all workflows				
11 workflow runs				
docs: update readme about CI CI Pipeline #11: Commit e251702 pushed by meerancor33	main	Success	Today at 1:42 AM	meerancor33
docs: update readme & fix: path database on db.py CI Pipeline #10: Commit 80f65d2 pushed by meerancor33	main	Success	Today at 1:29 AM	meerancor33
add: coverage test on docker CI Pipeline #9: Commit 981ac5b pushed by meerancor33	main	Success	Today at 1:15 AM	meerancor33
ci: add flake8's ignore CI Pipeline #8: Commit 2519155 pushed by meerancor33	main	Success	Today at 12:01 AM	meerancor33
add: health check details CI Pipeline #7: Commit cba90f1 pushed by meerancor33	main	Success	Dec 11, 11:59 PM GMT+7	meerancor33
add: health check endpoint and Docker healthcheck configuration CI Pipeline #6: Commit cb0dd8d pushed by meerancor33	main	Success	Dec 11, 11:48 PM GMT+7	meerancor33
ci: edit flake8's ignore CI Pipeline #5: Commit 71c2330 pushed by meerancor33	main	Success	Dec 11, 11:36 PM GMT+7	meerancor33
feat: add Dockerfile and improve project setup (requirements update, ... CI Pipeline #4: Commit 8567a21 pushed by meerancor33	main	Success	Dec 11, 11:31 PM GMT+7	meerancor33
ci: add github actions workflow for linting, tests, and docker build (3) CI Pipeline #3: Commit e95bb00 pushed by meerancor33	main	Success	Dec 11, 10:33 PM GMT+7	meerancor33
ci: add github actions workflow for linting, tests, and docker build CI Pipeline #2: Commit d5a6a5e pushed by meerancor33	main	Success	Dec 11, 10:30 PM GMT+7	meerancor33
ci: add github actions workflow for linting, tests, and docker build CI Pipeline #1: Commit e2110ad pushed by meerancor33	main	Success	Dec 11, 10:19 PM GMT+7	meerancor33

Gambar 3. Tampilan CI Pipeline Workflow Log

### C. Manfaat Integrasi Docker dalam CI/CD

Penggunaan Docker memastikan konsistensi lingkungan, sehingga *service* berjalan identik baik di mesin *developer*, dalam *pipeline* CI, maupun di lingkungan produksi. Docker juga menyediakan isolasi *dependency*, sehingga *service* tidak tergantung pada *library* atau versi Python yang terpasang di host. Integrasi dengan CI/CD menjadi lebih mudah karena *build* dan *testing container* dapat dijalankan secara otomatis dalam *pipeline* GitHub Workflow. Selain itu, Docker mendukung skenario TDD yang lebih realistik, karena *unit test* dijalankan di environment yang sama dengan produksi, termasuk akses ke database atau *service* lain yang dibutuhkan. Transparansi dan *monitoring* juga terjaga, karena status *pipeline* CI dapat langsung dipantau di tab “Actions”, memungkinkan *developer* segera menindaklanjuti jika terjadi *error* atau *test* yang gagal.

Dengan integrasi CI/CD dan Docker, pengembangan sistem *inventory control* menjadi lebih aman, otomatis, konsisten, dan siap untuk *deployment* berkelanjutan. Workflow memastikan setiap kode baru diuji, diverifikasi, dan dapat dijalankan di *container* produksi tanpa mengganggu fitur yang sudah ada, sekaligus memberikan indikator visual keberhasilan yang jelas di GitHub.

# DOKUMENTASI REQUEST & RESPONSE PAYLOAD

Dokumentasi ini menyajikan format *request* dan *response* untuk setiap endpoint pada *service Inventory Control*, mencakup peran Admin, Client/OHS, dan Manager. Tujuan utamanya adalah memberikan panduan jelas mengenai *payload* yang diterima dan dikembalikan oleh API, baik dalam kondisi berhasil maupun dalam skenario *error* atau *edge case*. Dengan adanya dokumentasi ini, setiap *endpoint* memiliki acuan standar yang konsisten, sehingga integrasi antar-modul maupun komunikasi dengan *client* pihak ketiga dapat dilakukan dengan lebih aman dan efisien.

## A. Authentication Endpoints

Register User	
Endpoint	POST /auth/register
Deskripsi	Mendaftarkan user baru dengan role tertentu (admin, manager, client).
Request Payload	{ "username": "johndoe", "password": "Secret123!", "full_name": "John Doe", "role": "admin" }
Response	(Success – HTTP 200) { "message": "User registered successfully." }  (Error – HTTP 400) { "detail": "Username already registered" }

Login User	
Endpoint	POST /auth/login
Deskripsi	Mengautentikasi user dan menghasilkan token JWT.
Request Payload	username=johndoe password=Secret123!
Response	(Success – HTTP 200) { "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...", "token_type": "bearer" }  (Error – HTTP 400) { "detail": "Incorrect username or password" }

	}
--	---

Logout User	
Endpoint	POST /auth/logout
Deskripsi	Melakukan invalidasi token JWT sehingga user tidak dapat menggunakannya lagi.
Request Header	Authorization: Bearer <access_token>
Response	<pre>(Success - HTTP 200) {     "message": "Logout successful. Token blacklisted." }  (Error - HTTP 401) {     "detail": "Could not validate credentials" }</pre>

## B. Health & Status

Health Check	
Endpoint	GET /health
Deskripsi	Memeriksa status service dan koneksi database. Endpoint ini digunakan untuk monitoring atau memastikan service berjalan dengan baik.
Request Payload	Tidak ada payload yang diperlukan.
Response	<pre>(Success - HTTP 200) {     "status": "ok",     "database": "up",     "timestamp": "2025-12-12T12:34:56.789Z" }  (Degraded - HTTP 200) {     "status": "degraded",     "database": "down",     "timestamp": "2025-12-12T12:35:12.345Z" }</pre>

## C. Admin Endpoints (Inventory Management)

Create Item	
Endpoint	POST /admin/items
Deskripsi	Menambahkan item baru ke inventory.

Request Payload	<pre>{     "sku": "A01",     "initial_qty": 100,     "uom": "pcs",     "min_qty": 10 }</pre>
Response	<p>(Success - HTTP 200)</p> <pre>{     "id": "item-uuid",     "sku": "A01",     "on_hand": 100,     "reserved": 0,     "available": 100,     "uom": "pcs",     "min_qty": 10,     "low_stock": false,     "reservations": [] }</pre> <p>(Error - Duplicate SKU, HTTP 400)</p> <pre>{     "detail": "SKU already exists" }</pre>

List All Items	
Endpoint	GET /admin/items
Deskripsi	Menampilkan daftar semua item di inventory.
Request Payload	Tidak ada payload yang diperlukan.
Response	<p>(Success - HTTP 200)</p> <pre>[     {         "id": "item-uuid",         "sku": "A01",         "on_hand": 100,         "reserved": 0,         "available": 100,         "uom": "pcs",         "min_qty": 10,         "low_stock": false,         "reservations": []     },     {         "id": "item-uuid2",         "sku": "B01",         "on_hand": 50,         "reserved": 5,         "available": 45,         "uom": "pcs",         "min_qty": 5,         "low_stock": false,         "reservations": []     } ]</pre>

	<pre>         }     ] </pre>
--	------------------------------

Get Item by SKU	
Endpoint	GET /admin/items/{sku}
Deskripsi	Mendapatkan detail item berdasarkan SKU.
Request Payload	Tidak ada payload yang diperlukan.
Response	<pre> (Success - HTTP 200) {     "id": "item-uuid",     "sku": "A01",     "on_hand": 100,     "reserved": 0,     "available": 100,     "uom": "pcs",     "min_qty": 10,     "low_stock": false,     "reservations": [] }  (Error - SKU not found, HTTP 404) {     "detail": "Item not found" } </pre>

Set Low Stock Threshold	
Endpoint	POST /admin/items/{sku}/threshold
Deskripsi	Mengatur nilai minimum stock (low stock threshold) untuk item tertentu.
Request Payload	<pre> {     "min_qty": 20 } </pre>
Response	<pre> (Success - HTTP 200) {     "id": "item-uuid",     "sku": "A01",     "on_hand": 100,     "reserved": 0,     "available": 100,     "uom": "pcs",     "min_qty": 20,     "low_stock": false,     "reservations": [] }  (Error - SKU not found, HTTP 404) { } </pre>

	<pre>         "detail": "Item not found"     } </pre>
--	-------------------------------------------------------

Adjust Stock Manually	
Endpoint	POST /admin/items/{sku}/adjust
Deskripsi	Menyesuaikan stock item secara manual, baik menambah maupun mengurangi.
Request Payload	<pre> {     "delta": -10,     "reason": "ADJ_NEG" } </pre>
Response	<p>(Success – HTTP 200)</p> <pre> {     "id": "item-uuid",     "sku": "A01",     "on_hand": 90,     "reserved": 0,     "available": 90,     "uom": "pcs",     "min_qty": 20,     "low_stock": false,     "reservations": [] } </pre> <p>(Error – Adjustment melebihi on-hand, HTTP 400)</p> <pre> {     "detail": "Cannot adjust stock below zero" } </pre>

List All Users	
Endpoint	GET /admin/users
Deskripsi	Menampilkan semua user yang terdaftar. Endpoint hanya bisa diakses oleh admin.
Request Payload	Tidak ada payload yang diperlukan.
Response	<p>(Success – HTTP 200)</p> <pre> [     {         "username": "admin01",         "full_name": "Admin One",         "role": "admin",         "disabled": false     },     {         "username": "manager01",         "full_name": "Manager One",         "role": "manager",         "disabled": true     } ] </pre>

```

        "disabled": false
    }
]

(Error - Unauthorized, HTTP 403)
{
    "detail": "Not enough permissions"
}

```

#### D. Client/OHS Endpoints (Stock Operations)

Get Item Availability	
Endpoint	GET /ohs/availability/{sku}
Deskripsi	Mendapatkan informasi ketersediaan stock untuk SKU tertentu.
Request Payload	Tidak ada payload yang diperlukan.
Response	<p>(Success - HTTP 200)</p> <pre>{     "sku": "A01",     "on_hand": 100,     "reserved": 20,     "available": 80,     "uom": "pcs",     "low_stock": false }</pre> <p>(Error - SKU not found, HTTP 404)</p> <pre>{     "detail": "Item not found" }</pre>

Increase Stock	
Endpoint	POST /ohs/{sku}/increase
Deskripsi	Menambah stock item tertentu.
Request Payload	<pre>{     "qty": 10,     "reason": "INBOUND" }</pre>
Response	<p>(Success - HTTP 200)</p> <pre>{     "id": "item-uuid",     "sku": "A01",     "on_hand": 110,     "reserved": 20,     "available": 90,     "uom": "pcs",     "min_qty": 10, }</pre>

```

        "low_stock": false,
        "reservations": []
    }

    (Error - Invalid SKU, HTTP 400)
{
    "detail": "Item not found"
}

```

Decrease Stock	
Endpoint	POST /ohs/{sku}/decrease
Deskripsi	Mengurangi stock item tertentu.
Request Payload	{         "qty": 15,         "reason": "CONSUME"     }
Response	(Success - HTTP 200)         {             "id": "item-uuid",             "sku": "A01",             "on_hand": 85,             "reserved": 20,             "available": 65,             "uom": "pcs",             "min_qty": 10,             "low_stock": false,             "reservations": []         }          (Error - Stock tidak cukup, HTTP 400)         {             "detail": "Not enough stock to decrease"         }

Reserve Stock	
Endpoint	POST /ohs/{sku}/reserve
Deskripsi	Membuat reservasi stock untuk order tertentu.
Request Payload	{         "order_id": "ORD123",         "qty": 5     }
Response	(Success - HTTP 200)         {             "id": "item-uuid",             "sku": "A01",             "on_hand": 85,         }

```

    "reserved": 25,
    "available": 60,
    "uom": "pcs",
    "min_qty": 10,
    "low_stock": false,
    "reservations": [
        {
            "id": "reservation-uuid",
            "order_id": "ORD123",
            "reserved_qty": 5
        }
    ]
}

(Error - Stock tidak cukup, HTTP 400)
{
    "detail": "Cannot reserve more than available stock"
}

```

Release Reservation	
Endpoint	POST /ohs/{sku}/release
Deskripsi	Melepaskan/ membatalkan reservation yang sudah dibuat.
Request Payload	{         "reservation_id": "reservation-uuid"     }
Response	(Success - HTTP 200) {     "id": "item-uuid",     "sku": "A01",     "on_hand": 85,     "reserved": 20,     "available": 65,     "uom": "pcs",     "min_qty": 10,     "low_stock": false,     "reservations": [] }  (Error - Reservation tidak ditemukan, HTTP 400) {     "detail": "Reservation not found" }

List Reservations	
Endpoint	GET /ohs/{sku}/reservations
Deskripsi	Menampilkan daftar semua reservasi untuk SKU tertentu.

Request Payload	<pre>{     "reservation_id": "reservation-uuid" }</pre>
Response	<p>(Success – HTTP 200)</p> <pre>[     {         "id": "reservation-uuid",         "order_id": "ORD123",         "reserved_qty": 5     },     {         "id": "reservation-uuid2",         "order_id": "ORD124",         "reserved_qty": 10     } ]</pre> <p>(Error – SKU tidak ditemukan, HTTP 404)</p> <pre>{     "detail": "Item not found" }</pre>

#### E. Manager Endpoints (Monitoring)

List Low Stock Items	
Endpoint	GET /manager/low-stock
Deskripsi	Menampilkan daftar item yang stoknya berada di bawah threshold minimum.
Request Payload	Tidak ada payload yang diperlukan.
Response	<p>(Success – HTTP 200)</p> <pre>[     {         "id": "item-uuid-1",         "sku": "A01",         "on_hand": 5,         "reserved": 2,         "available": 3,         "uom": "pcs",         "min_qty": 10,         "low_stock": true,         "reservations": [             {                 "id": "reservation-uuid-1",                 "order_id": "ORD001",                 "reserved_qty": 2             }         ]     },     {         "id": "item-uuid-2",         "sku": "B01",         "on_hand": 10,         "reserved": 0,         "available": 10,         "uom": "units",         "min_qty": 5,         "low_stock": false,         "reservations": []     } ]</pre>

```
        "on_hand": 2,
        "reserved": 0,
        "available": 2,
        "uom": "pcs",
        "min_qty": 5,
        "low_stock": true,
        "reservations": []
    }
]

(Success - Semua item stok aman, HTTP 200)
[]

(Error - Jika token tidak valid atau role bukan manager,
HTTP 403)
{
    "detail": "Not authorized"
}
```

# MENJALANKAN APLIKASI DAN LINGKUNGAN EKSEKUSI

Sebelum aplikasi dijalankan, dilakukan proses instalasi dan persiapan lingkungan kerja untuk memastikan seluruh dependency dapat berfungsi dengan baik. Proyek ini menggunakan Python dan menjalankan aplikasi melalui FastAPI dengan server ASGI Unicorn. Adapun langkah-langkah instalasi dan eksekusi adalah sebagai berikut.

## 1. Instalasi dan Persiapan Lingkungan

- 1) Clone repository

```
git clone https://github.com/meerancor33/II3160_TB_18223126_WMS.git  
cd II3160_TB_18223126_WMS
```

- 2) Buat virtual environment

```
python -m venv .venv
```

- 3) Aktifkan virtual environment

```
## PowerShell  
.venv\Scripts\Activate  
  
## Git Bash / MinGW  
source .venv/Scripts/activate  
  
## macOS / Linux  
source .venv/bin/activate  
  
## Jika berhasil, terminal akan menampilkan prefix:  
(.venv)
```

- 4) Install dependencies

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

## 2. Menjalankan Aplikasi

- a) Lokal

Setelah seluruh dependency terpasang, aplikasi dijalankan menggunakan perintah:

```
uvicorn src.main:app --reload
```

Jika berjalan dengan sukses, server akan menampilkan pesan:

```
Uvicorn running on http://127.0.0.1:8000  
Application startup complete.
```

Pada implementasi ini, ketika pengguna membuka URL <http://127.0.0.1:8000>, sistem secara otomatis melakukan redirect ke halaman dokumentasi API (Swagger UI) untuk memudahkan proses pengujian endpoint.

- b) Docker

Pertama bangun Docker Image terlebih dahulu menggunakan perintah:

```
docker compose up app --build
```

Jika berjalan dengan sukses, pada terminal akan muncul pesan:

```
inventory-app | INFO:      Started server process [1]
inventory-app | INFO:      Waiting for application startup.
inventory-app | WARNING:root:
=====
inventory-app | FastAPI server is running inside Docker
inventory-app | Open: http://localhost:8000/docs
inventory-app | Health Check: http://localhost:8000/health
inventory-app | Note: http://0.0.0.0:8000 TIDAK bisa dibuka dari browser
=====
inventory-app | INFO:      Application startup complete.
inventory-app | INFO:      Uvicorn running on http://0.0.0.0:8000 (Press
CTRL+C to quit)
```

Jika container berjalan dengan sukses, aplikasi dapat diakses melalui <http://localhost:8000>.

```
Ivana@LAPTOP-06GCRACU MINGW64 /d/II3160_TB_18223126_WMS (main)
$ docker compose up app --build
time="2025-12-12T10:20:20+07:00" level=warning msg="D:\\II3160_TB_18223126_WMS\\docker-compose.yml: the attribute 'version' is obsolete, it will be ignored, please remove it to avoid potential confusion"
[+] Building 3.0s (12/12) FINISHED
  => [internal] load local bake definitions
  => [internal] reading from stdin 524B
  => [internal] load build definition from Dockerfile
  => [internal] transferring dockerfile: 400B
  => [internal] load metadata for docker.io/library/python:3.11-slim
  => [internal] load .dockerrigore
  => [internal] load context: 2B
  => [1/5] FROM docker.io/library/python:3.11-slim@sha256:7cd0079a9bd8800c81632d65251048fc2848bf9afda542224b1b10e0cae45579
  => => resolve docker.io/library/python:3.11-slim@sha256:7cd0079a9bd8800c81632d65251048fc2848bf9afda542224b1b10e0cae45579
  => [internal] load build context
  => => transferring context: 10.04KB
  => [2/5] WORKDIR /app
  => CACHED [3/5] COPY requirements.txt .
  => CACHED [4/5] RUN pip install --no-cache-dir -r requirements.txt
  => CACHED [5/5] COPY . .
  => exporting to image
  => => exporting layers
  => => exporting manifest sha256:6250ade0cd8e45aa750b352bc18c5b56e7dfa02a2b7cb068ddd87ef9d85fbda5
  => => exporting config sha256:449d9b7a117bec4da2f0f19e4ab6779babad3efffab2cf1cd3b6c2729188b2
  => => exporting attestation manifest sha256:d0ee0940479ec59abff744051d93248a738080f194ed20a8166fb2a00c4
  => => exporting manifest list sha256:976ff66a504b1a84be330544decccd359ff3b401e62c8bb90b7d1f23f84f3864f9
  => => naming to docker.io/library/ii3160_tb_18223126_wms-app:latest
  => => unpacking to docker.io/library/ii3160_tb_18223126_wms-app:latest
  => => resolving provenance for metadata file
[+] Running 2/2
  ✓ ii3160_tb_18223126_wms-app Built
  ✓ Container inventory-app Recreated
Attaching to inventory-app
inventory-app | INFO:      Started server process [1]
inventory-app | INFO:      Waiting for application startup.
inventory-app | WARNING:root:
=====
inventory-app | FastAPI server is running inside Docker
inventory-app | Open: http://localhost:8000/docs
inventory-app | Health check: http://localhost:8000/health
inventory-app | Note: http://0.0.0.0:8000 TIDAK bisa dibuka dari browser
inventory-app | =====
inventory-app | INFO:      Application startup complete.
inventory-app | INFO:      Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
```

Gambar 4. Tampilan Terminal ketika Berhasil Membangun Docker & Container Berjalan

### 3. Dokumentasi & Pengujian API

FastAPI menyediakan dokumentasi otomatis yang dapat digunakan untuk menguji seluruh endpoint tanpa perlu menggunakan alat eksternal.

#### a) Swagger UI (Direkomendasikan)

Digunakan untuk melihat, mencoba, dan menguji seluruh endpoint API (GET, POST, DELETE, dsb).

```
http://127.0.0.1:8000/docs
http://localhost:8000/docs
```

b) ReDoc UI (Alternatif)

Menampilkan dokumentasi dalam format read-only dengan tampilan yang lebih terstruktur.

<http://127.0.0.1:8000/redoc>

<http://localhost:8000/redoc>

Melalui kedua dokumentasi ini, pengguna dapat memverifikasi seluruh fungsi API yang telah diimplementasikan, termasuk operasi admin gudang, inbound/outbound, reservasi, dan monitoring low stock.

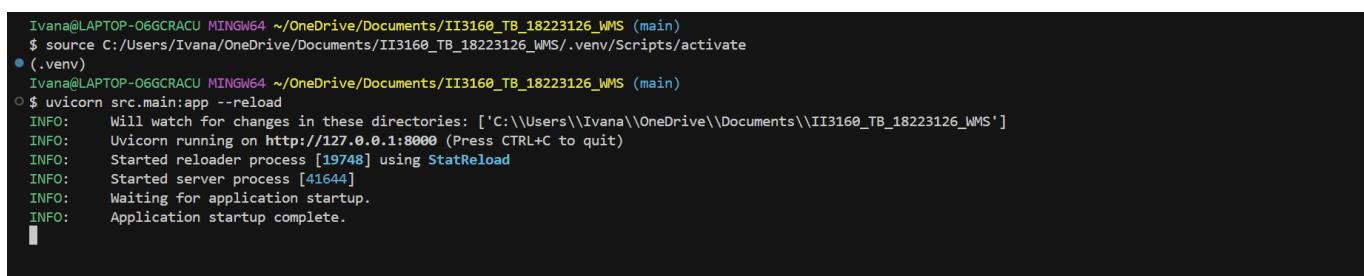
# PENGUJIAN API DENGAN SWAGGER UI & POSTMAN

Pengujian dilakukan untuk memastikan bahwa seluruh *endpoint* pada modul *Inventory Control* berfungsi sesuai dengan aturan bisnis yang telah ditetapkan pada domain model. Proses pengujian dilakukan menggunakan dua pendekatan, yaitu pengujian manual melalui Swagger UI dan Postman. Kedua alat ini memungkinkan pengembang atau penguji untuk memvalidasi request dan response secara langsung, memastikan bahwa alur kerja sistem berjalan sebagaimana mestinya.



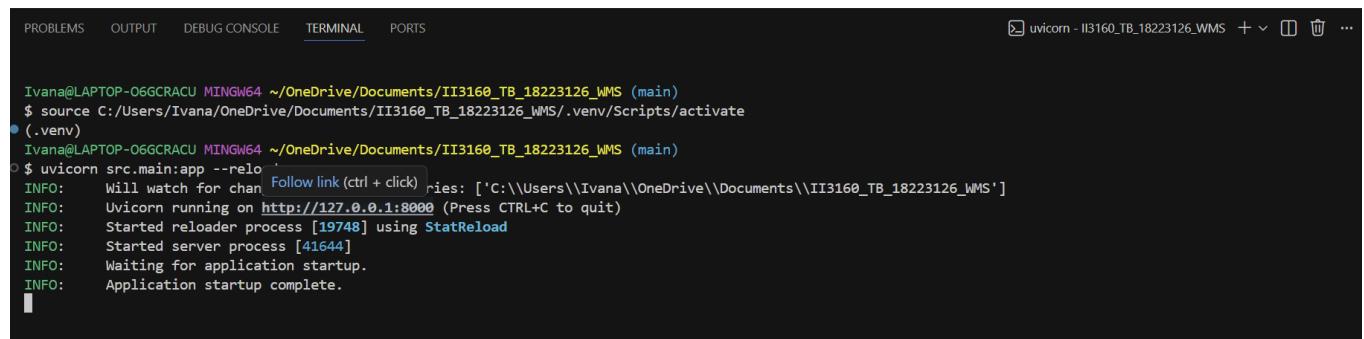
```
Ivana@LAPTOP-06GCRACU MINGW64 ~/OneDrive/Documents/II3160_TB_18223126_WMS (main)
$ source C:/Users/Ivana/OneDrive/Documents/II3160_TB_18223126_WMS/.venv/Scripts/activate
● (.venv)
Ivana@LAPTOP-06GCRACU MINGW64 ~/OneDrive/Documents/II3160_TB_18223126_WMS (main)
$ uvicorn src.main:app --reload
```

Gambar 5. Langkah Pertama untuk Menjalankan Aplikasi



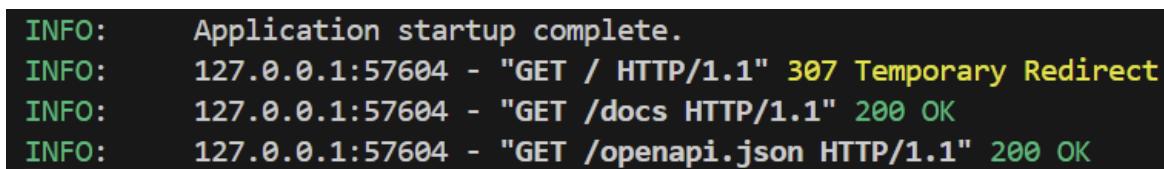
```
Ivana@LAPTOP-06GCRACU MINGW64 ~/OneDrive/Documents/II3160_TB_18223126_WMS (main)
$ source C:/Users/Ivana/OneDrive/Documents/II3160_TB_18223126_WMS/.venv/Scripts/activate
● (.venv)
Ivana@LAPTOP-06GCRACU MINGW64 ~/OneDrive/Documents/II3160_TB_18223126_WMS (main)
$ uvicorn src.main:app --reload
INFO: Will watch for changes in these directories: ['C:\\\\Users\\\\Ivana\\\\OneDrive\\\\Documents\\\\II3160_TB_18223126_WMS']
INFO: Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO: Started reloader process [19748] using StatReload
INFO: Started server process [41644]
INFO: Waiting for application startup.
INFO: Application startup complete.
```

Gambar 6. Tampilan ketika Aplikasi Berhasil Berjalan



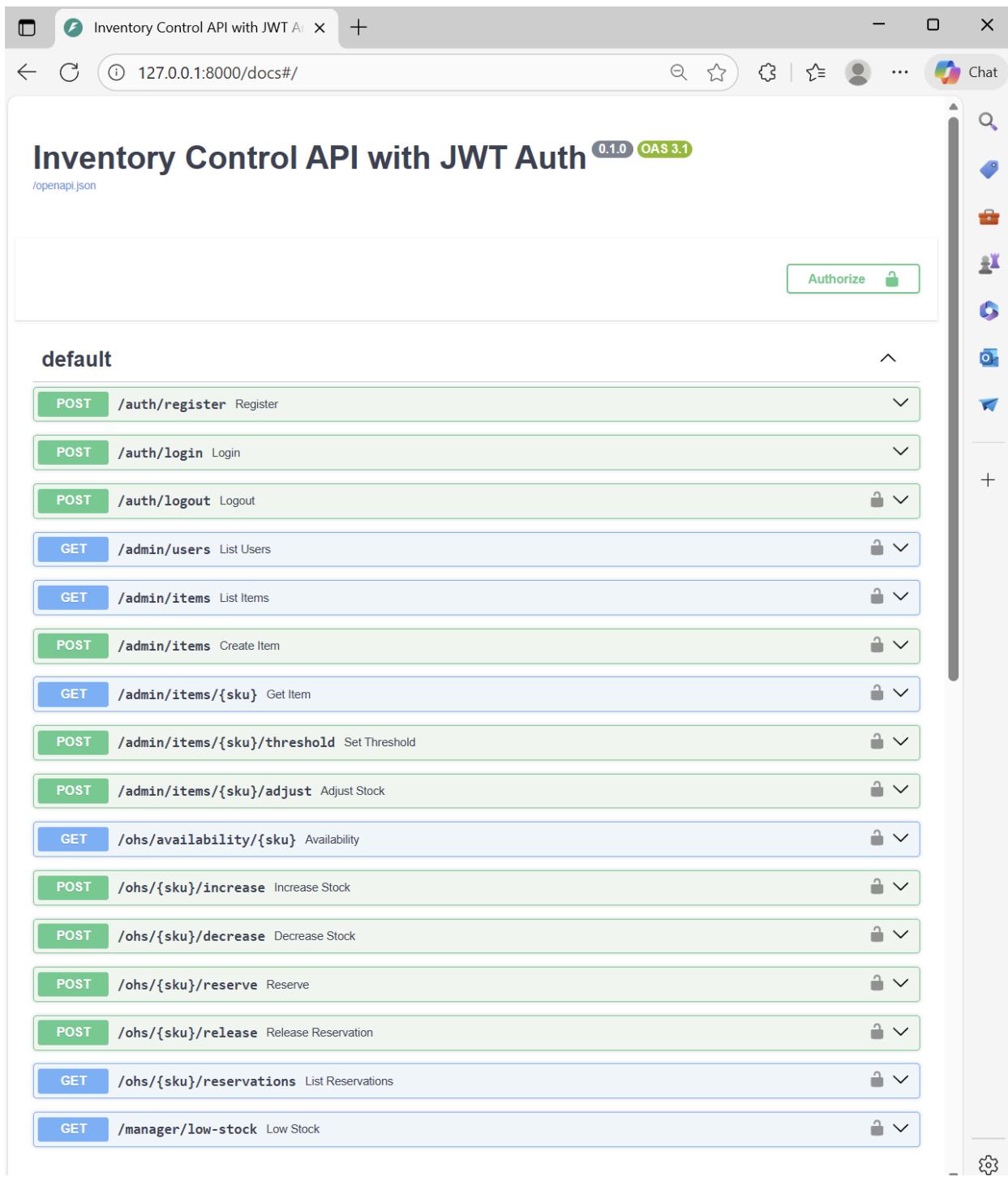
```
Ivana@LAPTOP-06GCRACU MINGW64 ~/OneDrive/Documents/II3160_TB_18223126_WMS (main)
$ source C:/Users/Ivana/OneDrive/Documents/II3160_TB_18223126_WMS/.venv/Scripts/activate
● (.venv)
Ivana@LAPTOP-06GCRACU MINGW64 ~/OneDrive/Documents/II3160_TB_18223126_WMS (main)
$ uvicorn src.main:app --reload
INFO: Will watch for changes [Follow link (ctrl + click)] ries: ['C:\\\\Users\\\\Ivana\\\\OneDrive\\\\Documents\\\\II3160_TB_18223126_WMS']
INFO: Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO: Started reloader process [19748] using StatReload
INFO: Started server process [41644]
INFO: Waiting for application startup.
INFO: Application startup complete.
```

Gambar 7. Tampilan Tautan untuk Redirect ke Swagger UI



```
INFO: Application startup complete.
INFO: 127.0.0.1:57604 - "GET / HTTP/1.1" 307 Temporary Redirect
INFO: 127.0.0.1:57604 - "GET /docs HTTP/1.1" 200 OK
INFO: 127.0.0.1:57604 - "GET /openapi.json HTTP/1.1" 200 OK
```

Gambar 8. Tampilan Ketika Fitur Redirect Berhasil Berjalan dan Swagger UI Berhasil Dimuat



Gambar 9. Tampilan Swagger UI Inventory Control

## A. Pengujian Menggunakan Swagger UI

Swagger UI merupakan fitur bawaan FastAPI yang menyediakan antarmuka interaktif untuk melihat dan mencoba seluruh endpoint. Melalui halaman ini, pengguna dapat mengirim request secara langsung, mengisi request body, dan mengamati response yang dihasilkan.

## Registrasi Pengguna

POST /auth/register

### Tampilan awal:

The screenshot shows the initial view of the API documentation for the `POST /auth/register` endpoint. It includes sections for `Parameters` (none), `Request body` (required, type `application/json`), and `Responses`. The responses section details two cases: a successful `200` response with a `string` media type, and a validation error `422` response with a complex JSON schema for `detail`.

### Tampilan Try It Out:

The screenshot shows the `Try It Out` interface for the `POST /auth/register` endpoint. It features a form to edit the `Request body` (type `application/json`) with a schema editor and an `Execute` button.

Tampilan Edit Request Body:

*Register Admin:*

**Request body** required

[Edit Value](#) | Schema

```
{  
    "username": "User123",  
    "password": "123User",  
    "full_name": "Admin Gudang",  
    "role": "admin"  
}
```

*Register Manager:*

**Request body** required

[Edit Value](#) | Schema

```
{  
    "username": "User231",  
    "password": "231User",  
    "full_name": "Manager Gudang",  
    "role": "manager"  
}
```

## Register Client/OHS:

Request body **required**

Edit Value | Schema

```
{  
    "username": "User312",  
    "password": "312User",  
    "full_name": "Client1",  
    "role": "client"  
}
```

## Tampilan Setelah Execute:

### Responses

#### Curl

```
curl -X 'POST' \  
  'http://127.0.0.1:8000/auth/register' \  
  -H 'accept: application/json' \  
  -H 'Content-Type: application/json' \  
  -d '{  
    "username": "User123",  
    "password": "123User",  
    "full_name": "Admin Gudang",  
    "role": "admin"  
}'
```

#### Request URL

```
http://127.0.0.1:8000/auth/register
```

#### Server response

Code	Details
200	Response body

```
{  
    "message": "User registered successfully."  
}
```

Response headers  
`content-length: 43  
content-type: application/json  
date: Mon, 01 Dec 2025 22:04:35 GMT  
server: uvicorn`

## Tampilan jika Mencoba Register dengan Username yang Sama:

### Responses

#### Curl

```
curl -X 'POST' \  
  'http://127.0.0.1:8000/auth/register' \  
  -H 'accept: application/json' \  
  -H 'Content-Type: application/json' \  
  -d '{  
    "username": "User123",  
    "password": "123User",  
    "full_name": "Admin",  
    "role": "admin"  
}'
```

#### Request URL

```
http://127.0.0.1:8000/auth/register
```

#### Server response

Code	Details
400 <small>Undocumented</small>	Error: Bad Request

Response body  
`{  
 "detail": "Username already registered"  
}`

Response headers  
`content-length: 49  
content-type: application/json  
date: Mon, 01 Dec 2025 22:05:33 GMT  
server: uvicorn`

## Login Pengguna

POST /auth/login

### Tampilan awal:

POST /auth/login Login

Parameters

No parameters

Request body required

application/x-www-form-urlencoded

```
grant_type
string | (string |
null)
pattern: password

username * required
string

password * required
string

scope
string

client_id
string | (string |
null)

client_secret
string | (string |
null)
```

Responses

Code	Description	Links
200	Successful Response	No links
	Media type: application/json	
	Controls Accept header.	
	Example Value   Schema	
	{ "access_token": "string", "token_type": "string" }	
422	Validation Error	No links
	Media type: application/json	
	Example Value   Schema	
	{ "detail": [ { "loc": [ "string", "string", "string", "string" ], "msg": "string", "type": "string" } ] }	

### Tampilan Try It Out:

POST /auth/login Login

Parameters

No parameters

Request body required

application/x-www-form-urlencoded

```
grant_type
string | (string |
null)
pattern: password
 Send empty value

username * required
string
 Send empty value

password * required
string
 Send empty value

scope
string
 Send empty value

client_id
string | (string |
null)
 Send empty value

client_secret
string | (string |
null)
 Send empty value
```

Execute

## Tampilan Edit Request Body:

### Login Admin:

Request body required

**grant\_type**  
string | (string | null)  
pattern: password

password

Send empty value

**username** \* required  
string

User123

**password** \* required  
string

123User

**scope**  
string

scope

Send empty value

**client\_id**  
string | (string | null)

string

Send empty value

**client\_secret**  
string | (string | null)

string

Send empty value

## Register Manager:

Request body required

**grant\_type**  
string | (string | null)  
pattern: password

password

Send empty value

**username** \* required  
string

User231

**password** \* required  
string

231User

**scope**  
string

scope

Send empty value

**client\_id**  
string | (string | null)

string

Send empty value

**client\_secret**  
string | (string | null)

string

Send empty value

## Register Client/OHS:

Request body required

grant_type string   (string   null) pattern: password	<input type="text" value="password"/> <input checked="" type="checkbox"/> Send empty value
username <small>* required</small> string	<input type="text" value="User312"/>
password <small>* required</small> string	<input type="text" value="312User"/>
scope string	<input type="text" value="scope"/> <input checked="" type="checkbox"/> Send empty value
client_id string   (string   null)	<input type="text" value="string"/> <input type="checkbox"/> Send empty value
client_secret string   (string   null)	<input type="text" value="string"/> <input type="checkbox"/> Send empty value

## Tampilan Setelah Execute:

Responses

Curl

```
curl -X 'POST' '\
http://127.0.0.1:8000/auth/login' \
-H 'Content-Type: application/x-www-form-urlencoded' \
-d 'grant_type=password&username=User312&password=312User&scope=scope&client_id=string&client_secret=string'
```

Request URL

http://127.0.0.1:8000/auth/login

Server response

Code	Details
200	<p>Response body</p> <pre>{ "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpVVCJ9.eyJzdWIiOiJVc2VyRmFtciI6IC0yZTg1MjEwNjkwNjIyMzY0NjQ5MDY5IQ.sPKuvWfFfHIXoualNVugS0531vLZkUTrauzERlk3k", "token_type": "bearer" }</pre> <p>Response headers</p> <pre>content-length: 188 content-type: application/json date: Mon, 01 Dec 2025 22:17:48 GMT server: unicorn</pre>

Untuk Bisa Menjalankan *Endpoint* sesuai *Role*, User harus melakukan “*Authorize*”:

The screenshot illustrates the process of authorizing an API call through the Swagger UI. A modal dialog titled "Available authorizations" is displayed, prompting the user to enter credentials for OAuth2PasswordBearer (OAuth2, password). The "username" field is populated with "User123" and the "password" field contains "\*\*\*\*\*". The "client\_id" and "client\_secret" fields are also present. Below this, a "Save your password?" modal is overlaid, asking if the user wants to save their password to their Microsoft account. The background shows a list of API endpoints for an "Inventory Control API with JW...".

## Logout Pengguna

POST /auth/logout

### Tampilan awal:

The screenshot shows the initial view of the Logout endpoint. It includes a 'POST /auth/logout Logout' button, a 'Parameters' section (No parameters), and a 'Responses' section. The 'Responses' section shows a 200 status code with a successful response message and a schema of 'string'. A 'Try it out' button is located in the top right corner.

### Tampilan Try It Out:

The screenshot shows the 'Try It Out' interface for the Logout endpoint. It features a large blue 'Execute' button at the top. Below it are sections for 'Parameters' (No parameters) and 'Responses'. The 'Responses' section is identical to the initial view, showing a 200 status code with a successful response message and a schema of 'string'. A 'Cancel' button is located in the top right corner.

### Tampilan Edit Request Body: Tidak ada body

### Tampilan Setelah Execute:

The screenshot shows the results of executing the Logout endpoint. It includes a 'Request URL' field with 'curl -X POST \ http://127.0.0.1:8000/auth/logout \ -H "Content-Type: application/json" \ -H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInRS..."' and a 'Server response' field with the URL 'http://127.0.0.1:8000/auth/logout'. The 'Responses' section shows a 200 status code with a response body containing the message 'Logout successful. Token blacklisted.' and a response header with 'Content-Length: 51', 'Content-Type: application/json', 'Date: Mon, 01 Dec 2025 22:55:39 GMT', and 'Server: unicorn'. A 'Download' button is located next to the response body.

## Membuat item baru

POST /admin/items

### Tampilan awal:

The screenshot shows the API documentation for the `POST /admin/items` endpoint. It includes sections for Parameters, Request body, Example Value, Responses, and Examples.

**Parameters:** No parameters.

**Request body required:** application/json

**Example Value | Schema:**

```
{
  "sku": "SKU-001",
  "initial_qty": 0,
  "uom": "pcs",
  "min_qty": 0
}
```

**Responses:**

Code	Description	Links
200	Successful Response	No links
422	Validation Error	No links

**Successful Response Example Value | Schema:**

```
{
  "id": "string",
  "name": "string",
  "on_hand": 0,
  "available": 0,
  "uom": "string",
  "min": "string",
  "max": "string",
  "low_stock": true,
  "reservations": []
}
```

**Validation Error Example Value | Schema:**

```
{
  "detail": [
    {
      "loc": [
        "string",
        "string"
      ],
      "msg": "string",
      "type": "string"
    }
  ]
}
```

### Tampilan Try It Out:

The screenshot shows the "Try It Out" interface for the `POST /admin/items` endpoint. It includes sections for Parameters, Request body, and an Execute button.

**Parameters:** No parameters.

**Request body required:** application/json

**Edit Value | Schema:**

```
{
  "sku": "SKU-001",
  "initial_qty": 0,
  "uom": "pcs",
  "min_qty": 0
}
```

**Execute**

### Tampilan Edit Request Body:

The screenshot shows the "Edit Request Body" dialog. It displays the required schema for the request body.

**Request body required**

**Edit Value | Schema**

```
{
  "sku": "SKU-011",
  "initial_qty": 1000,
  "uom": "pcs",
  "min_qty": 10
}
```

## Tampilan Setelah Execute:

Responses

Curl

```
curl -X POST \
http://127.0.0.1:8000/admin/items \
-H 'accept: application/json' \
-H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpVVCJ9.eyJzdWIiOiJvYmVzZW1lLzIwcm9zZS16TnfkbmluIiwidXh0IjoxNzExMjM0NTxxQ.vd3av_oB7fjprOhyhPBOtNxgyAIDwh1cOy4SV_3K6A' \
-d '{ \
    "sku": "SKU-011", \
    "initial_qty": 1000, \
    "user": "pcx", \
    "min_qty": 10 \
}'
```

Request URL  
<http://127.0.0.1:8000/admin/items>

Server response

Code Details

200

Response body

```
[{"id": "c444bf8c-ca7e-47c2-9e83-9aa0e467bdec", "sku": "SKU-011", "initial_qty": 1000, "reserved": 0, "available": 1000, "user": "pcx", "min_qty": 10}, {"id": "c444bf8c-ca7e-47c2-9e83-9aa0e467bdec", "sku": "SKU-011", "initial_qty": 1000, "reserved": 0, "available": 1000, "user": "pcx", "min_qty": 10}], "reservations": []}
```

Response headers

```
content-length: 167
content-type: application/json
date: Mon, 01 Dec 2025 22:26:53 GMT
server: unicorn
```

[Download](#)

## Tampilan jika Mencoba Menambah Item dengan ID SKU yang Sama:

Responses

Curl

```
curl -X POST \
http://127.0.0.1:8000/admin/items \
-H 'accept: application/json' \
-H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpVVCJ9.eyJzdWIiOiJvYmVzZW1lLzIwcm9zZS16TnfkbmluIiwidXh0IjoxNzExMjM0NTxxQ.vd3av_oB7fjprOhyhPBOtNxgyAIDwh1cOy4SV_3K6A' \
-d '{ \
    "sku": "SKU-011", \
    "initial_qty": 1000, \
    "user": "pcx", \
    "min_qty": 0 \
}'
```

Request URL  
<http://127.0.0.1:8000/admin/items>

Server response

Code Details

400 Error: Bad Request

Undocumented Response body

```
{"detail": "SKU already exists"}
```

Response headers

```
content-length: 31
content-type: application/json
date: Mon, 01 Dec 2025 22:26:53 GMT
server: unicorn
```

[Download](#)

## Tampilan jika User dengan Role Client atau Manager yang Melakukan Penambahan Item:

POST /admin/items Create item

Parameters

No parameters

Request body required

application/json

Edit Value | Schema

```
{"sku": "SKU-011", "initial_qty": 1000, "user": "pcx", "min_qty": 10}
```

Execute Clear

Responses

Curl

```
curl -X POST \
http://127.0.0.1:8000/admin/items \
-H 'accept: application/json' \
-H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpVVCJ9.eyJzdWIiOiJvYmVzZW1lLzIwcm9zZS16TnfkbmluIiwidXh0IjoxNzExMjM0NTxxQ.yok30HhC1dijgeQ_Qursh7226_lIUoJ5WfIQ88Efu' \
-d '{ \
    "sku": "SKU-011", \
    "initial_qty": 1000, \
    "user": "pcx", \
    "min_qty": 10 \
}'
```

Request URL  
<http://127.0.0.1:8000/admin/items>

Server response

Code Details

403 Error: Forbidden

Undocumented Response body

```
{"detail": "Access denied"}
```

Response headers

```
content-length: 26
content-type: application/json
date: Mon, 01 Dec 2025 22:42:27 GMT
server: unicorn
```

[Download](#)

## Melihat daftar item

GET /admin/items

### Tampilan awal:

The screenshot shows the 'List Items' endpoint. It includes a 'Parameters' section with 'No parameters', a 'Responses' section with a 'Successful Response' (status 200) example in application/json format, and a large redacted JSON response body.

### Tampilan Try It Out:

The screenshot shows the 'Try It Out' interface for the GET /admin/items endpoint. It displays the curl command and an 'Execute' button.

### Tampilan Edit Request Body: Tidak ada body

### Tampilan Setelah Execute:

The screenshot shows the response after executing the GET /admin/items endpoint. It displays a detailed JSON response with two items, each having attributes like id, sku, on\_hand, reserved\_qty, available\_qty, min\_qty, max\_qty, uom, and low\_stock.

### Tampilan jika User dengan Role Client atau Manager yang Melakukan Pengecekan List Item:

Responses

```
Curl
curl -X 'GET' \
http://127.0.0.1:8000/admin/items' \
-H 'accept: application/json' \
-H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCIpKzC39.eyJzdWIiOiJ7c2VjMzYtZWJlYiwiexp0a2StIiV4cC10Tc2NDYyPjJUAH0.yGKS80nC15jpmC_Qzrsh7268_11052MF1Q8BfU'
Request URL
http://127.0.0.1:8000/admin/items
Server response
Code Details
403 Error: Forbidden
Undocumented Response body
{
    "detail": "Access denied"
}
Response headers
content-length: 26
content-type: application/json
date: Mon, 01 Mar 2021 22:43:27 GMT
server: unicorn
-
```

[Download](#)

## Melihat detail item berdasarkan SKU

GET /admin/items/{sku}

### Tampilan awal:

GET /admin/items/{sku} Get Item

Parameters

Name	Description
sku * required	sku (path)

Responses

Code	Description	Links
200	Successful Response	No links

Media type: application/json  
Controls Accept header.

Example Value: Schema

```
{
    "id": "string",
    "sku": "string",
    "name": "string",
    "reserved": 0,
    "available": 0,
    "unit": "string",
    "min_qty": 0,
    "low_stock": true,
    "reservations": []
}
```

### Tampilan Try It Out:

GET /admin/items/{sku} Get Item

Parameters

Name	Description
sku * required	sku (path)

Execute

### Tampilan Edit Request Body:

Parameters

Name	Description
sku * required	SKU-01 (path)

Execute

### Tampilan Setelah Execute:

Responses

```
Curl
curl -X 'GET' \
http://127.0.0.1:8000/admin/items/SKU-01' \
-H 'accept: application/json' \
-H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCIpKzC39.eyJzdWIiOiJ7c2VjMzYtZWJlYiwiexp0a2StIiV4cC10Tc2NDYyPjJUAH0.yGKS80nC15jpmC_Qzrsh7268_11052MF1Q8BfU'
Request URL
http://127.0.0.1:8000/admin/items/SKU-01
Server response
Code Details
200 Response body
{
    "id": "string",
    "sku": "SKU-01",
    "name": "string",
    "reserved": 0,
    "available": 0,
    "unit": "string",
    "min_qty": 0,
    "low_stock": true,
    "reservations": []
}
Response headers
content-length: 167
content-type: application/json
date: Mon, 01 Mar 2021 22:43:27 GMT
server: unicorn
-
```

## Melakukan peningkatan stok

POST /ohs/{sku}/increase

### Tampilan awal:

The screenshot shows the API documentation for the `/ohs/{sku}/increase` endpoint. It includes sections for Parameters, Request body, and Responses.

**Parameters**

Name	Description
sku * required	string (path)

**Request body** required

Example Value : Schema

```
{ "qty": 6, "reason": "INBOUND" }
```

**Responses**

Code	Description	Links
200	Successful Response	No links

Media type: application/json

Controls Accept header

Example Value : Schema

```
{ "id": "string", "on_hand": 6, "reserved": 0, "stock": 6, "use": "string", "low_stock": true, "reservations": [] }
```

### Tampilan Try It Out:

The screenshot shows the `/ohs/{sku}/increase` endpoint in the Try It Out section. It has fields for Parameters and Request body.

**Parameters**

Name	Description
sku * required	string (path)

**Request body** required

Edit Value : Schema

```
{ "qty": 6, "reason": "INBOUND" }
```

Execute

### Tampilan Edit Request Body:

The screenshot shows the `/ohs/{sku}/increase` endpoint in the Edit Request Body section. It displays the filled-in parameters and request body.

**Parameters**

Name	Description
sku * required	SKU-011

**Request body** required

Edit Value | Schema

```
{ "qty": 10, "reason": "INBOUND" }
```

### Tampilan Setelah Execute:

Responses

```
Curl
curl -X "POST" \
  "http://127.0.0.1:8000/ohs/SKU-011/increase" \
  -H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInRS...eyJzdW1O12Vc2Y9MzYiLiwi...ZST6ImRsaWVudC1sIav4cC18MTc2NDYzRjUwM0B.yGK5800C515jg..._Quzshe7Z6B_110US2MFQ8BtFU" \
  -d '{"qty": 10, "reason": "INBOUND"}'
```

Request URL  
<http://127.0.0.1:8000/ohs/SKU-011/increase>

Server response

Code	Details
200	<p>Response body</p> <pre>{   "id": "44466fb8-ca7e-47c2-9e83-9adbe467b0ec",   "sku": "SKU-011",   "on_hand": 2110,   "reserved": 0,   "available": 2110,   "use": "string",   "min_qty": 30,   "low_stock": false,   "reservations": [] }</pre> <p>Response headers</p> <pre>content-length: 167 content-type: application/json date: Mon, 20 Dec 2025 22:44:42 GMT server: uvicorn</pre>

[Download](#)

### Melakukan pengurangan stok

POST /ohs/{sku}/decrease

### Tampilan awal:

POST /ohs/{sku}/decrease Decrease Stock

Parameters

Name	Description
sku <small>required</small>	string (path)

Request body required

application/json

Example Value : Schema

```
{
  "qty": 0,
  "reason": "CONSUME"
}
```

Responses

Code	Description	Links
200	Successful Response	No links

Media type  
application/json

Content Accept header

Example Value : Schema

```
{
  "id": "string",
  "on_hand": 0,
  "reserved": 0,
  "available": 0,
  "use": "string",
  "min_qty": 0,
  "low_stock": true,
  "reservations": []
}
```

### Tampilan Try It Out:

POST /ohs/{sku}/decrease Decrease Stock

Parameters

Name	Description
sku <small>required</small>	string (path)

Request body required

application/json

Edit Value : Schema

```
{
  "qty": 0,
  "reason": "CONSUME"
}
```

Execute

### Tampilan Edit Request Body:

**Parameters**

Name	Description
sku * required	SKU-011 string (path)

**Request body** required

```
{
  "qty": 60,
  "reason": "CONSUME"
}
```

### Tampilan Setelah Execute:

**Responses**

Curl

```
curl -X POST http://127.0.0.1:8000/ohs/SKU-011/decrease \
-H 'accept: application/json' \
-H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCIKIpWC39.y7mM1013Vc2Y9y6lyLiwieZST67mRaWudC16Tav4cCTIGHTc2NDYpJjwA40.yGS80nCS16jpmC_Qurzshe7268_110US2WFQ8BxFU' \
-H 'Content-Type: application/json' \
-d '{
  "qty": 60,
  "reason": "CONSUME"
}'
```

Request URL

http://127.0.0.1:8000/ohs/SKU-011/decrease

Server response

Code Details

200 Response body

```
{
  "id": "c4468bf8-ca7e-47c1-9c83-9aa0e467b9ec",
  "on hand": 2000,
  "reserved": 0,
  "qty": 6000,
  " uom": "pcs",
  "type": "Stock",
  "low_stock": false,
  "reservations": []
}
```

Response headers

```
content-length: 167
date: Mon, 01 Dec 2025 22:45:39 GMT
server: unicorn
```

### Melakukan reservasi stok

POST /ohs/{sku}/reserve

### Tampilan awal:

**POST /ohs/{sku}/reserve Reserve Stock**

**Parameters**

Name	Description
sku * required	sku string (path)

**Request body** required

```
{
  "order_id": "string",
  "qty": 0
}
```

**Responses**

Code	Description	Links
200	Successful Response	No links

Media type: application/json

Example Value | Schema

```
{
  "id": "string",
  "sku": "string",
  "on hand": 0,
  "reserved": 0,
  "available": 0
}
```

## Tampilan Try It Out:

POST /ohs/{sku}/reserve Reserve Stock

Cancel

Parameters

Name	Description
sku <small>* required</small>	SKU

Request body required

application/json

Edit Value | Schema

```
{ "order_id": "string", "qty": 0 }
```

Execute

## Tampilan Edit Request Body:

Parameters

Name	Description
sku <small>* required</small>	SKU-011

Request body required

Edit Value | Schema

```
{ "order_id": "User312_1", "qty": 150 }
```

## Tampilan Setelah Execute:

Responses

Curl

```
curl -X 'POST' 'http://127.0.0.1:8000/ohs/SKU-011/reserve' \
-H 'Content-Type: application/json' \
-H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9eyJmIjoiMjAxNjMwOTk4NzQyNzEwMSIsInV4cC10MTc2NDYzPjUwMjB0_yGK580hC516jpmc_Qurzhe7Z6B_110052WF1Q8FtFU' \
-d '{
    "order_id": "User312_1",
    "qty": 150
}'
```

Request URL

http://127.0.0.1:8000/ohs/SKU-011/reserve

Server response

Code Details

200 Response body

```
[
  {
    "id": "e4446bf8-ca7e-47c1-9e83-9aafe467bdcc",
    "sku": "SKU-011",
    "available_qty": 150,
    "reserved_qty": 150,
    "min_qty": 1,
    "max_qty": 1000,
    "locked": false,
    "reservations": [
      {
        "id": "9a214580-0d4f-4595-866b-79415161247e",
        "order_id": "User312_1",
        "reserved_qty": 150
      }
    ]
]
```

Download

Response headers

```
content-length: 256
content-type: application/json
date: Mon, 01 Dec 2025 22:47:05 GMT
server: unicorn
```

## Melihat daftar reservasi stok

GET /ohs/{sku}/reservations

### Tampilan awal:

The screenshot shows the API documentation for the '/ohs/{sku}/reservations' endpoint. It includes sections for 'Parameters', 'Responses', and 'Example Value | Schema'. The 'Parameters' section shows a required 'sku' parameter of type string (path). The 'Responses' section shows two examples: a successful response (200) with a schema of 'string' and a validation error (422) with a schema of 'object' containing 'detail' and 'loc' fields.

### Tampilan Try It Out:

The screenshot shows the 'Try It Out' interface for the '/ohs/{sku}/reservations' endpoint. It displays the same 'Parameters' and 'Responses' sections as the initial view, but with a prominent 'Execute' button at the bottom.

### Tampilan Edit Request Body:

The screenshot shows the 'Edit Request Body' interface for the '/ohs/{sku}/reservations' endpoint. It shows the 'Parameters' section with a 'sku' parameter set to 'SKU-011'. There is also an 'Execute' button at the bottom.

### Tampilan Setelah Execute: (saat belum ada reservasi)

The screenshot shows the results of executing the '/ohs/{sku}/reservations' endpoint with 'sku' set to 'SKU-012'. The 'Responses' section shows a curl command and a request to 'http://127.0.0.1:8000/ohs/SKU-012/reservations'. The 'Server response' section shows a 200 status code with an empty JSON response body and the following headers:  
content-length: 2  
content-type: application/json  
date: Mon, 01 Dec 2025 21:46:38 GMT  
server: uvicorn

**(saat sudah ada reservasi):**

Responses

Curl

```
curl -X 'GET' \
'http://127.0.0.1:8080/ohs/SKU-B11/reservations' \
-H 'Content-Type: application/json' \
-H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVC9.yJzdwIi0l3vC2VyNzlyIiwicm9sZSI6ICt0NDYzMjAwMAw0.y0K58NC3dIjne_Qurshet7Z68_110U52WFQ8HrtU'
```

Request URL

<http://127.0.0.1:8080/ohs/SKU-B11/reservations>

Server response

Code	Details
200	<p>Response body</p> <pre>[   {     "id": "8421458bd-8d4f-4595-866b-79a15161247a",     "order_id": "User312_1",     "reserved_qty": 100   } ]</pre> <p>Response headers</p> <pre>content-length: 89 content-type: application/json date: Mon, 05 Dec 2025 21:48:11 GMT server: uvicorn</pre>

[Download](#)

## Membatalkan reservasi

POST /ohs/{sku}/release

**Tampilan awal:**

POST /ohs/{sku}/release Release Reservation

Parameters

Name	Description
sku <small>required</small>	string (path)

Request body required

Example Value | Schema

```
{
  "reservation_id": "string"
}
```

Responses

Code	Description	Links
200	Successful Response	No links

Media type: application/json  
Controls Accept header.

Example Value | Schema

```
{
  "id": "string",
  "order_id": "string",
  "on_hand": 0,
  "reserved_qty": 100,
  "available": 0,
  "user": "string",
  "unit": "string",
  "low_stock": true,
  "Reservations": []
}
```

**Tampilan Try It Out:**

POST /ohs/{sku}/release Release Reservation

Parameters

Name	Description
SKU <small>required</small>	string (path)

Request body required

Edit Value | Schema

```
{
  "reservation_id": "string"
}
```

Execute

## Tampilan Edit Request Body:

### Parameters

Name	Description
sku * required string (path)	SKU-011

### Request body required

#### Edit Value | Schema

```
{
  "reservation_id": "0421450d-0d4f-4595-8d6b-79415161247e"
}
```

## Tampilan Setelah Execute:

### Responses

```
Curl
curl -X 'POST' \
  'http://127.0.0.1:8000/ohs/SKU-011/release' \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVC9...eyJzdWIiOiJvc2Vycm9sZTAiInVudC1sJmV4cC10Tc2NDY2RjUmM0...yGK380nC516jgnC__Quzshe7268_110052WF1Q8tFU' \
  -d '{
    "reservation_id": "0421450d-0d4f-4595-8d6b-79415161247e"
}'
```

Request URL  
<http://127.0.0.1:8000/ohs/SKU-011/release>

Server response

Code Details

200 Response body

```
[
  {
    "id": "0421450d-0d4f-4595-8d6b-79415161247e",
    "sku": "SKU-011",
    "on_hand": 2056,
    "available": 2056,
    "low": "yes",
    "low_qty": 1000,
    "low_stock": false,
    "reservations": []
  }
]
```

[Download](#)

Response headers

```
content-length: 367
content-type: application/json
date: Mon, 01 Dec 2025 22:58:20 GMT
server: unicorn
```

## Tampilan saat cek reservasi stok SKU-001: sudah kosong

### Responses

```
Curl
curl -X 'POST' \
  'http://127.0.0.1:8000/ohs/SKU-011/release' \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVC9...eyJzdWIiOiJvc2Vycm9sZTAiInVudC1sJmV4cC10Tc2NDY2RjUmM0...yGK380nC516jgnC__Quzshe7268_110052WF1Q8tFU' \
  -d '{
    "reservation_id": "0421450d-0d4f-4595-8d6b-79415161247e"
}'
```

Request URL  
<http://127.0.0.1:8000/ohs/SKU-011/release>

Server response

Code Details

400 Error: Bad Request

Indicates that the request was received by the server but contained syntax errors or was outside the range of valid operations.

```
{
  "detail": "Reservation not found"
}
```

[Download](#)

Response headers

```
content-length: 36
content-type: application/json
date: Mon, 01 Dec 2025 22:58:39 GMT
server: unicorn
```

### Parameters

[Cancel](#)

Name	Description
sku * required string (path)	SKU-011

Execute

Clear

### Responses

```
Curl
curl -X 'GET' \
  'http://127.0.0.1:8000/ohs/SKU-011/reservations' \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVC9...eyJzdWIiOiJvc2Vycm9sZTAiInVudC1sJmV4cC10Tc2NDY2RjUmM0...yGK380nC516jgnC__Quzshe7268_110052WF1Q8tFU'
```

Request URL  
<http://127.0.0.1:8000/ohs/SKU-011/reservations>

Server response

Code Details

200 Response body

```
[ ]
```

[Download](#)

Response headers

```
content-length: 2
content-type: application/json
date: Mon, 01 Dec 2025 22:58:39 GMT
server: unicorn
```

## Melihat ketersediaan stok

GET /ohs/availability/{sku}

### Tampilan awal:

The screenshot shows the API documentation for the 'Get Availability' endpoint. It includes sections for 'Parameters' (with a required 'sku' parameter), 'Responses' (200: Successful Response, 422: Validation Error), and a 'Try it out' button.

### Tampilan Try It Out:

The screenshot shows the 'Try It Out' interface for the 'Get Availability' endpoint. It displays the same parameters and responses as the initial view, with an additional 'Execute' button at the bottom.

### Tampilan Edit Request Body:

The screenshot shows the 'Edit Request Body' interface for the 'Get Availability' endpoint. It shows a single parameter 'sku' with the value 'SKU-011' and an 'Execute' button.

### Tampilan Setelah Execute:

The screenshot shows the results of the executed request. It includes a 'Curl' command, a 'Request URL' (http://127.0.0.1:8000/ohs/availability/SKU-011), a 'Server response' section with a 200 status code, and a 'Details' section showing the response body and headers. The response body is a JSON object with fields like 'sku', 'on\_hand', 'reserved', 'available', 'unit', and 'low\_stock'. The response headers include 'Content-Length', 'Content-Type: application/json', 'Date', and 'Server'.

## Melihat daftar low stock

GET /manager/low-stock

### Tampilan awal:

The screenshot shows the 'Responses' section for a successful 200 response. It displays a JSON schema example:

```
[{"id": "string", "sku": "string", "on_hand": 0, "reserved": 0, "available": 0, "min_qty": 0, "low_stock": true, "reservations": []}]
```

### Tampilan Try It Out:

The screenshot shows the 'Execute' button being clicked.

### Tampilan Edit Request Body: Tidak ada body

### Tampilan Setelah Execute:

The screenshot shows the 'Server response' section for a 200 response. It includes a curl command, request URL, and response body:

```
curl -X 'GET' 'http://127.0.0.1:8000/manager/low-stock' \
-H 'accept: application/json'
```

Request URL: <http://127.0.0.1:8000/manager/low-stock>

Server response

Response body:

```
[{"id": "string", "sku": "string", "on_hand": 0, "reserved": 0, "available": 0, "min_qty": 0, "low_stock": true, "reservations": []}]
```

### Tampilan Setelah SKU-001 berada di < Min Qty:

The screenshot shows the 'Server response' section for a 200 response. It includes a curl command, request URL, and response body:

```
curl -X 'GET' 'http://127.0.0.1:8000/manager/low-stock' \
-H 'accept: application/json'
```

Request URL: <http://127.0.0.1:8000/manager/low-stock>

Server response

Response body:

```
[{"id": "string", "sku": "SKU-001", "on_hand": 0, "reserved": 0, "available": 0, "min_qty": 10, "low_stock": true, "reservations": []}]
```

## Mengatur Threshold Baru

POST  
/admin/items/{sku}/threshold

### Tampilan awal:

The screenshot shows the API documentation for the `POST /admin/items/{sku}/threshold Set Threshold` endpoint. It includes sections for Parameters, Request body, and Responses.

**Parameters:** A parameter named `sku` is listed as required, with a type of string (path).

**Request body:** Required. Example value: `{"min_qty": 0}`. Schema: `{ "min_qty": 0 }`.

**Responses:** 200 - Successful Response. Media type: application/json. Content accept header: application/json. Example value: Schema: `{ "id": "string", "sku": "string", "name": "string", "reserved": 0, "low_stock": 0, "low": "string", "now": "string", "low_stock": true, "reservations": [] }`.

### Tampilan Try It Out:

The screenshot shows the `Try It Out` interface for the `POST /admin/items/{sku}/threshold Set Threshold` endpoint. It has fields for Parameters and Request body, and a large text area for the response.

**Parameters:** A parameter named `sku` is listed as required, with a type of string (path).

**Request body:** Required. Edit Value: `{"min_qty": 0}`.

**Responses:** Execute button.

### Tampilan Edit Request Body:

The screenshot shows the `Edit Request Body` interface for the `POST /admin/items/{sku}/threshold Set Threshold` endpoint.

**Parameters:** A parameter named `sku` is listed as required, with a type of string (path). Value: SKU-011.

**Request body:** Required. Edit Value | Schema: `{ "min_qty": 30 }`.

### Tampilan Setelah Execute:

Responses

```
Curl
curl -X 'POST' \
http://127.0.0.1:8000/admin/items/SKU-011/threshold' \
-H 'accept: application/json' \
-H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpVCS9yZjdhM101YzZ9YH1xTzTzLwz5t29WfjoxNzWJh6NTkxQ.v63av_oB7fjprOhyuP901XgylDvh1cGy4SV_900A' \
-d '{
    "min_qty": 30
}'
```

Request URL  
<http://127.0.0.1:8000/admin/items/SKU-011/threshold>

Server response

Code	Details
200	<p>Response body</p> <pre>{     "id": "c4dd6bf8-ca7e-47c2-9eb3-9aa0e407bdec",     "sku": "SKU-011",     "name": "Product A",     "reserved": 0,     "available": 1000,     "unit": "PC",     "min_qty": 30,     "max_qty": 1000,     "low_stock": false,     "reservations": [] }</pre> <p>Response headers</p> <pre>content-length: 167 content-type: application/json date: Mon, 22 Mar 2021 13:53:53 GMT server: unicorn</pre>

[Download](#)

### Mengatur Stock

POST  
</admin/items/{sku}/adjust>

### Tampilan awal:

POST /admin/items/{sku}/adjust - Adjust Stock

Parameters

Name	Description
sku * required	string (path)

Request body required

application/json

Example Value | Schema

```
{
    "delta": 0,
    "reason": "ADJUST"
}
```

Responses

Code	Description	Links
200	Successful Response	No links
422	Validation Error	No links

application/json

Example Value | Schema

```
{
    "detail": [
        {
            "loc": [
                "sku"
            ],
            "msg": "SKU must be provided"
        }
    ]
}
```

### Tampilan Try It Out:

Parameters

Name	Description
sku * required	string (path)

Request body required

application/json

Edit Value | Schema

```
{
    "delta": 0,
    "reason": "ADJUST"
}
```

Execute Clear

	<p><b>Tampilan Edit Request Body:</b></p> <table border="1"> <thead> <tr> <th>Name</th><th>Description</th></tr> </thead> <tbody> <tr> <td>sku * required string (path)</td><td>SKU-011</td></tr> </tbody> </table> <pre>{   "delta": 1100,   "reason": "ADJUST" }</pre>	Name	Description	sku * required string (path)	SKU-011
Name	Description				
sku * required string (path)	SKU-011				
	<p><b>Tampilan Setelah Execute:</b></p> <p>Curl</p> <pre>curl -X POST \   http://127.0.0.1:8000/admin/items/SKU-011/adjust \   -H 'Accept: application/json' \   -H 'Authorization: Bearer eyJhbGciOiIuTi1lNisIn8cCIRkpkVCJ9.eyJzdWIiOiIxMjYwNjdlbmVyc2h5b24iLCJpc3MiOiJsb2dpbiIsIi1XQmJjZG93ZWJpZDhnbG9yA1DwNlcjAySV_908A' \   -H 'Content-Type: json' \   -d '{     "delta": 1100,     "reason": "ADJUST"   }'</pre> <p>Request URL</p> <p>http://127.0.0.1:8000/admin/items/SKU-011/adjust</p> <p>Server response</p> <table border="1"> <thead> <tr> <th>Code</th> <th>Details</th> </tr> </thead> <tbody> <tr> <td>200</td> <td> <p>Response body</p> <pre>{   "id": "c44d8fb8-ca7e-47c2-9eb3-9aa8e467bdec",   "sku": "SKU-011",   "hand": 1,   "reserved": 0,   "delta": 1100,   "user": "pca",   "min_qty": 100,   "track": false,   "reservations": [] }</pre> <p>Download</p> <p>Response headers</p> <pre>content-length: 167 content-type: application/json date: wed, 08 dec 2023 22:35:04 gmt server: uvicorn</pre> </td> </tr> </tbody> </table>	Code	Details	200	<p>Response body</p> <pre>{   "id": "c44d8fb8-ca7e-47c2-9eb3-9aa8e467bdec",   "sku": "SKU-011",   "hand": 1,   "reserved": 0,   "delta": 1100,   "user": "pca",   "min_qty": 100,   "track": false,   "reservations": [] }</pre> <p>Download</p> <p>Response headers</p> <pre>content-length: 167 content-type: application/json date: wed, 08 dec 2023 22:35:04 gmt server: uvicorn</pre>
Code	Details				
200	<p>Response body</p> <pre>{   "id": "c44d8fb8-ca7e-47c2-9eb3-9aa8e467bdec",   "sku": "SKU-011",   "hand": 1,   "reserved": 0,   "delta": 1100,   "user": "pca",   "min_qty": 100,   "track": false,   "reservations": [] }</pre> <p>Download</p> <p>Response headers</p> <pre>content-length: 167 content-type: application/json date: wed, 08 dec 2023 22:35:04 gmt server: uvicorn</pre>				

Swagger UI secara otomatis menampilkan struktur request dan response, lengkap dengan tipe data, contoh input, dan dokumentasi parameter, sehingga sangat membantu dalam memastikan bahwa setiap endpoint bekerja sesuai spesifikasi.

## Responses

Curl

```
curl -X 'GET' \
'http://127.0.0.1:8000/admin/items' \
-H 'Accept: application/json' \
-H 'Authorization: Bearer eyJhbGciOiJIUzI1NiTsTrScC76TpXVC39-eyJzdldlOjIwC2V9yNTIx2iwcm9sZT6TmFkhiIuTiwZXhuTjoxNzYBkjMaTKkfQ.v63av_v0R7FjprDhyHuPBOTDQgyAIIvhicGy4SV_9KRA'
```

Request URL

```
http://127.0.0.1:8000/admin/items
```

Server response

Code	Details
200	<p>Response body</p> <pre>{   "reserved": 0,   "available": 196,   " uom": "pcs",   "min_qty": 10,   "low_stock": false,   "reservations": [] }, {   "id": "c44d6bf8-ca7e-47c2-9e83-9aa0e467b9ec",   "sku": "SKU-011",   "on_hand": 2100,   "reserved": 0,   "available": 2100,   " uom": "pcs",   "min_qty": 30,   "low_stock": false,   "reservations": [] }, {   "id": "13313658-0817-41ce-b93d-0a554538f173",   "sku": "SKU-012",   "on_hand": 2000,   "reserved": 0,   "available": 2000,   " uom": "pcs",   "min_qty": 20,   "low_stock": false,   "reservations": [] } ]</pre> <p>Download</p> <p>Response headers</p> <pre>content-length: 919 content-type: application/json date: Mon, 01 Dec 2025 22:39:33 GMT server: unicorn</pre>

Gambar 10. Tampilan Kondisi akhir Swagger Setelah Testing API di Swagger

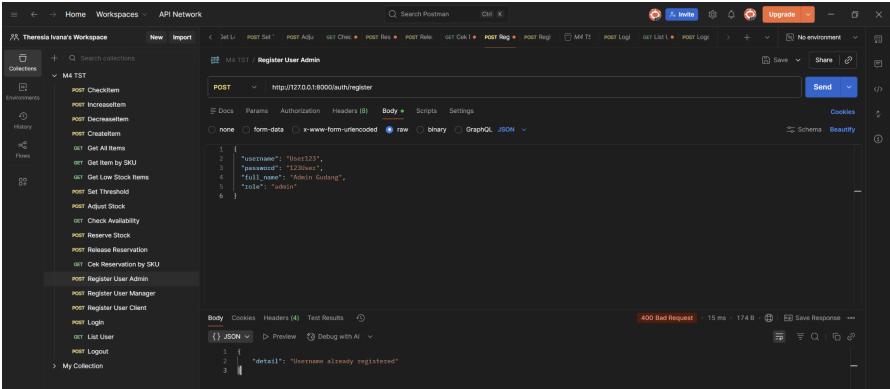
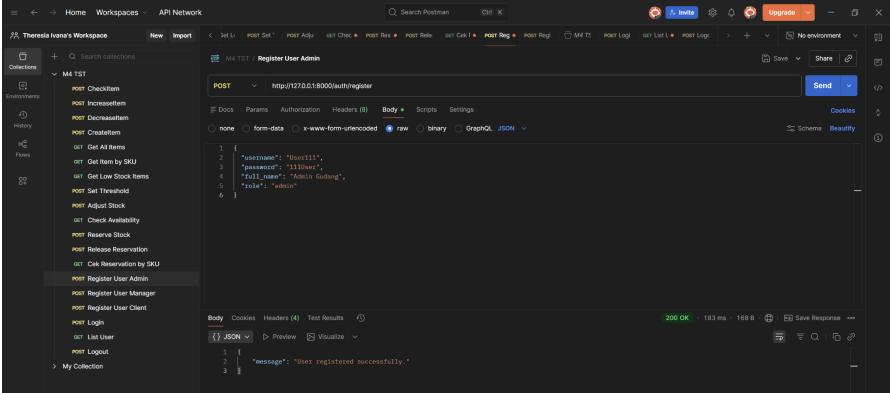
The screenshot shows the Postman application interface. On the left, there's a sidebar with 'Collections' (M4 TST), 'Environments', 'History', and 'Favorites'. The main area shows a 'GET Get All Items' request for 'http://127.0.0.1:8000/admin/items'. The 'Authorization' tab is selected, showing 'Bearer Token' with a placeholder 'Token' and a redacted token value. The 'Headers' tab shows 'Content-Type: application/json'. The 'Body' tab is collapsed. The 'JSON' tab is selected, displaying the API response. The response is a JSON array of two items, each representing a product with fields like id, sku, uom, min\_qty, low\_stock, and reservations. The first item has an id of 'c44d6bf8-ca7e-47c2-9e83-9aa0e467b9ec', sku 'SKU-011', uom 'pcs', min\_qty 10, and low\_stock false. The second item has an id of '13313658-0817-41ce-b93d-0a554538f173', sku 'SKU-012', uom 'pcs', min\_qty 20, and low\_stock false.

```
{
  "id": "c44d6bf8-ca7e-47c2-9e83-9aa0e467b9ec",
  "sku": "SKU-011",
  "on_hand": 2100,
  "reserved": 0,
  "available": 2100,
  " uom": "pcs",
  "min_qty": 30,
  "low_stock": false,
  "reservations": []
},
{
  "id": "13313658-0817-41ce-b93d-0a554538f173",
  "sku": "SKU-012",
  "on_hand": 2000,
  "reserved": 0,
  "available": 2000,
  " uom": "pcs",
  "min_qty": 20,
  "low_stock": false,
  "reservations": []
}
```

Gambar 11. Tampilan Kondisi awal Postman Setelah Testing API di Swagger

## B. Pengujian Menggunakan Postman

Selain Swagger UI, pengujian juga dilakukan menggunakan Postman untuk memastikan API dapat diakses dan digunakan melalui protokol HTTP standar tanpa ketergantungan pada antarmuka FastAPI.

Registrasi Pengguna	
POST /auth/register	<p><b>Tampilan jika Mencoba Register dengan Username yang sudah Dibuat di Swagger:</b></p>  <p>The screenshot shows the Postman interface with a POST request to <code>http://127.0.0.1:8000/auth/register</code>. The request body contains the following JSON:</p> <pre>1   { "username": "test123", 2   "password": "123456", 3   "full_name": "Admin Gading", 4   "role": "admin" 5   } 6   </pre> <p>The response status is 400 Bad Request, with the message: "detail": "Username already registered".</p> <p><b>Tampilan jika Mencoba Register dengan Username Baru:</b></p>  <p>The screenshot shows the Postman interface with a POST request to <code>http://127.0.0.1:8000/auth/register</code>. The request body contains the following JSON:</p> <pre>1   { "username": "test1111", 2   "password": "111111", 3   "full_name": "Adelin Gading", 4   "role": "admin" 5   } 6   </pre> <p>The response status is 200 OK, with the message: "message": "User registered successfully."</p>

## Login Pengguna

POST /auth/login

Tampilan jika Berhasil Login:

The screenshot shows a successful POST request to the '/auth/login' endpoint. The response status is 200 OK with a response time of 175 ms. The response body is a JSON object containing an access token and a token type.

```
1 {
2   "access_token": "eyJhbGciOiJIUzI1NiwiZW5kc1giXVCIJ...",
3   "token_type": "bearer"
4 }
```

Tampilan jika Tidak Berhasil Login (Username Belum Teregister):

The screenshot shows an unsuccessful POST request to the '/auth/login' endpoint. The response status is 400 Bad Request with a response time of 6 ms. The response body is a JSON object with a 'detail' field indicating an incorrect username or password.

```
1 [
2   "detail": "Incorrect username or password"
3 ]
```

## Logout Pengguna

POST /auth/logout

The screenshot shows a POST request to the '/auth/logout' endpoint. The response status is 200 OK with a response time of 176 ms. The response body is a JSON object with a 'message' field indicating a successful logout.

```
1 {
2   "message": "Logout successful. Token blacklisted."
3 }
```

## Membuat item baru

POST /admin/items

### Pengaturan Autorization:

The screenshot shows the Postman interface for a POST request to 'http://127.0.0.1:8000/admin/items'. In the 'Authorization' tab, 'Bearer Token' is selected from the dropdown. A placeholder 'Token' is present in the input field, with a note below stating: 'The authorization header will be automatically generated when you send the request. Learn more about [Bearer Token](#) authorization.'

### Body Request:

The screenshot shows the Postman interface for the same POST request. In the 'Body' tab, the raw JSON body is displayed as follows:

```
[{"sku": "SKU-005", "on_hand": 300, "use": "pc", "win_qty": 10}, {"sku": "SKU-003", "on_hand": 300, "use": "pc", "win_qty": 10}, {"sku": "SKU-001", "on_hand": 300, "use": "pc", "win_qty": 10}, {"sku": "SKU-002", "on_hand": 300, "use": "pc", "win_qty": 10}, {"sku": "SKU-004", "on_hand": 300, "use": "pc", "win_qty": 10}]
```

## Melihat daftar item

GET /admin/items

### Pengaturan Autorization:

This screenshot is identical to the one above, showing the Postman interface for a POST request to 'http://127.0.0.1:8000/admin/items'. It highlights the 'Authorization' tab with 'Bearer Token' selected.

### Body Request:

The screenshot shows the Postman interface for the same GET request. In the 'Body' tab, the raw JSON body is displayed as follows:

```
[{"sku": "SKU-005", "on_hand": 300, "use": "pc", "win_qty": 10}, {"sku": "SKU-003", "on_hand": 300, "use": "pc", "win_qty": 10}, {"sku": "SKU-001", "on_hand": 300, "use": "pc", "win_qty": 10}, {"sku": "SKU-002", "on_hand": 300, "use": "pc", "win_qty": 10}, {"sku": "SKU-004", "on_hand": 300, "use": "pc", "win_qty": 10}]
```

```

[{"id": "144d60e8-ca7e-47c2-9e83-9a0de467b7ec", "sku": "SKU-011", "on_hand": 2659, "reserved": 0, "available": 2659, "last_qty": 30, "line_stock": false, "reservations": []}, {"id": "13313608-0017-41ce-b93d-0a5453f173", "sku": "SKU-012", "on_hand": 2659, "reserved": 0, "available": 2659, "last_qty": 20, "line_stock": false, "reservations": []}]

```

## Melihat detail item berdasarkan SKU

GET /admin/items/{sku}

### Pengaturan Autorization:

### Tampilan Setelah Execute SKU-011:

```

[{"id": "144d60e8-ca7e-47c2-9e83-9a0de467b7ec", "sku": "SKU-011", "on_hand": 2659, "reserved": 0, "available": 2659, "last_qty": 30, "line_stock": false, "reservations": []}]

```

### Tampilan Setelah Execute SKU-014:

The screenshot shows the Postman interface with a collection named 'M4 TST'. A test case 'GET Item by SKU' is selected. The URL is set to `http://127.0.0.1:8000/admin/items/SKU-014`. The 'Authorization' header is set to 'Bearer Token'. The response status is 404 Not Found, and the body contains the JSON object: 

```
[{"detail": "Item not found"}]
```

## Melakukan peningkatan stok

POST /ohs/{sku}/increase

### Pengaturan Autorization:

The screenshot shows the Postman interface with a collection named 'M4 TST'. A test case 'IncreaseItem' is selected. The URL is set to `http://127.0.0.1:8000/ohs/SKU-012/increase`. The 'Authorization' header is set to 'Bearer Token'. The response status is 200 OK, and the body contains the JSON object: 

```
[{"id": "11111111-0000-401c-893d-0e55438f7731", "sku": "SKU-012", "in_hand": 2659, "type": "P", "available": 2566, "num": "pct", "high": 2659, "low_stock": false, "reservations": []}]
```

### Body Request:

Tampilan Setelah Execute Increase 50 Qty untuk SKU-012:

The screenshot shows the Postman interface with a collection named 'Theresa Ivanis Workspace'. A test case 'IncreaseItem' is selected. The URL is set to `http://127.0.0.1:8000/ohs/SKU-012/increase`. The 'Authorization' header is set to 'Bearer Token'. The body is set to `{"qty": 50}`. The response status is 200 OK, and the body contains the JSON object: 

```
[{"id": "11111111-0000-401c-893d-0e55438f7731", "sku": "SKU-012", "in_hand": 2659, "type": "P", "available": 2566, "num": "pct", "high": 2659, "low_stock": false, "reservations": []}]
```

## Melakukan pengurangan stok

POST /ohs/{sku}/decrease

Tampilan Setelah Execute Decrease 150 Qty untuk SKU-012:

The screenshot shows a Postman interface with a successful POST request to `http://127.0.0.1:8000/ohs/SKU-012/decrease`. The response body is a JSON object with the following data:

```

1 |   {
2 |     "id": "12345678-0017-41ce-9936-8e55453ff173",
3 |     "sku": "SKU-012",
4 |     "on_hand": 1990,
5 |     "reserved": 0,
6 |     "low_stock": 0,
7 |     "unit": "pc",
8 |     "min_qty": 20,
9 |     "low_stock": false,
10|     "reservations": []
11|

```

## Melakukan reservasi stok

POST /ohs/{sku}/reserve

Tampilan Setelah Execute Reserve SKU-012:

The screenshot shows a Postman interface with a successful POST request to `http://127.0.0.1:8000/ohs/SKU-012/reserve`. The response body is a JSON object with the following data:

```

1 |   {
2 |     "order_id": "0001 Client1",
3 |     "qty": 40
4 |
5|

```

Below the main response, there is another JSON object representing the reservation details:

```

1 |   [
2 |     {
3 |       "id": "a5c0000e-1bav-4x30-9914-fbded3d49a0",
4 |       "order_id": "0001 Client1",
5 |       "reserved_qty": 40
6 |
7 |     }
8 |
9 |

```

## Melihat daftar reservasi stok

GET /ohs/{sku}/reservations

Tampilan Setelah Execute Reservations SKU-012:

The screenshot shows a Postman interface with a successful GET request to `http://127.0.0.1:8000/ohs/SKU-012/reservations`. The response body is a JSON object with the following data:

```

1 |   [
2 |     {
3 |       "id": "a5c0000e-1bav-4x30-9914-fbded3d49a0",
4 |       "order_id": "0001 Client1",
5 |       "reserved_qty": 40
6 |
7 |     }
8 |
9 |

```

## Tampilan Setelah Execute Reservations SKU-013:

The screenshot shows the Postman interface with a collection named 'M4 TST'. A GET request is made to 'http://127.0.0.1:8000/ohs/SKU-013/reservations'. The response status is 200 OK, with a response time of 6 ms and a body size of 126 B. The response body is empty, indicated by a single brace '}'.

## Tampilan Setelah Execute Release Reservations SKU-012:

The screenshot shows the Postman interface with a collection named 'M4 TST'. A GET request is made to 'http://127.0.0.1:8000/ohs/SKU-012/reservations'. The response status is 200 OK, with a response time of 7 ms and a body size of 126 B. The response body is empty, indicated by a single brace '}'.

## Membatalkan reservasi

POST /ohs/{sku}/release

## Tampilan Setelah Execute Release Reservations SKU-012:

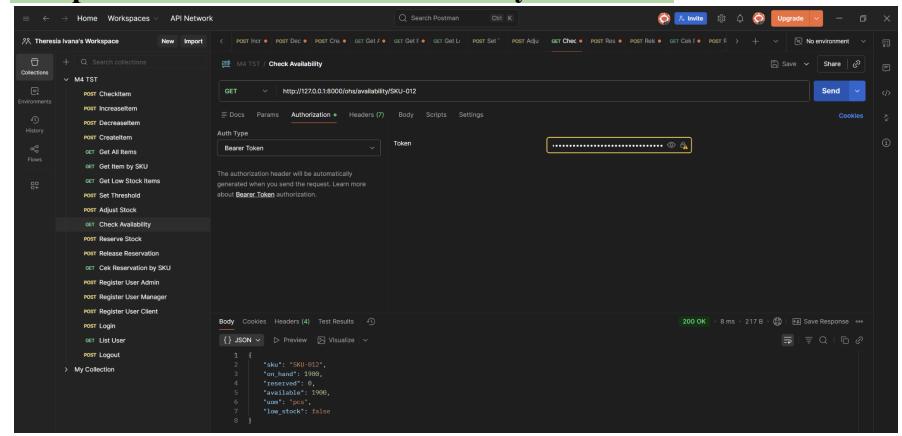
The screenshot shows the Postman interface with a collection named 'M4 TST'. A POST request is made to 'http://127.0.0.1:8000/ohs/SKU-012/release'. The response status is 200 OK, with a response time of 15 ms and a body size of 293 B. The response body is a JSON object containing reservation details:

```
1 | {
2 |   "id": "13313630-0817-41ce-93d-0a554538f173",
3 |   "item_id": "10000000000000000000000000000000",
4 |   "on_hand": 100,
5 |   "reserved": 0,
6 |   "available": 100,
7 |   "low": null,
8 |   "min_qty": 20,
9 |   "low_stock": false,
10|   "reservations": []
11| }
```

## Melihat ketersediaan stok

GET /ohs/availability/{sku}

### Tampilan Setelah Execute Availability SKU-012:

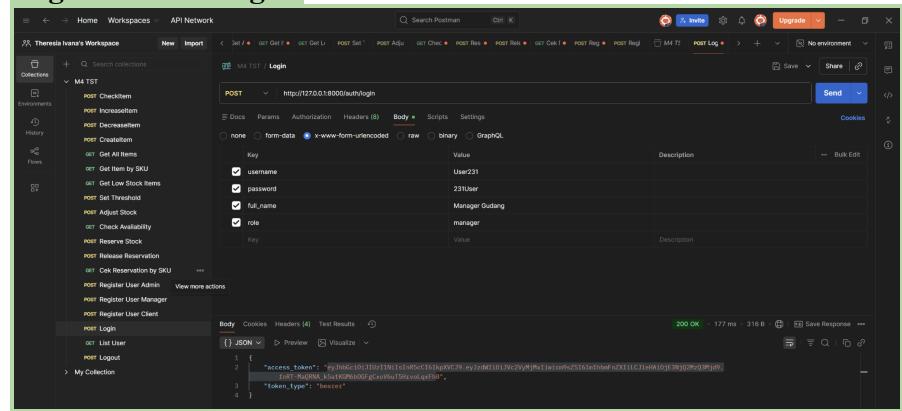


```
1 | {
2 |   "id": "012",
3 |   "ohs_id": 1999,
4 |   "reserved": 0,
5 |   "available": 1999,
6 |   "low": 0,
7 |   "low_stock": false
8 | }
```

## Melihat daftar low stock

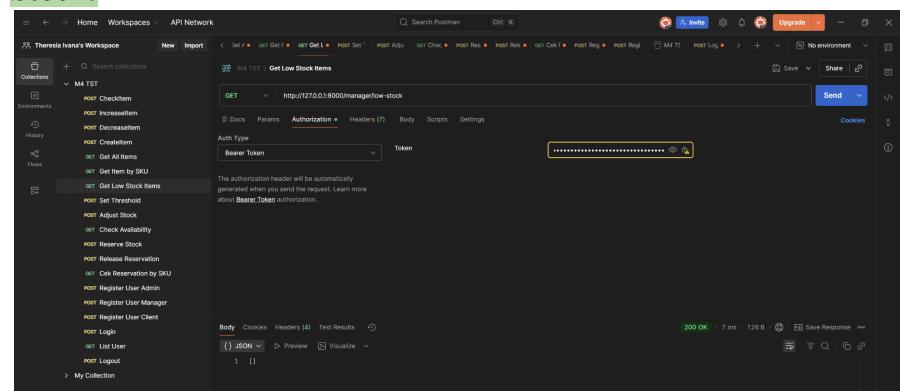
GET /manager/low-stock

### Login Role Manager:



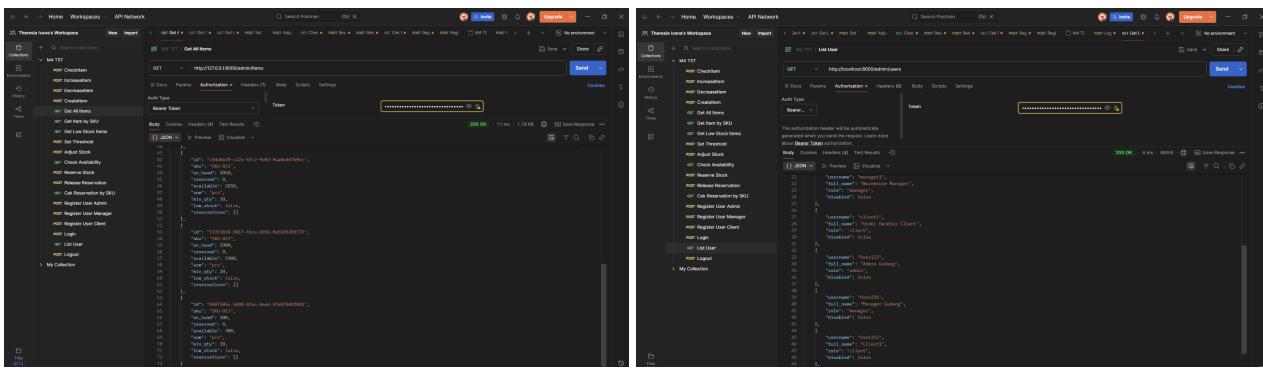
```
POST http://127.0.0.1:8000/auth/login
{
  "username": "User231",
  "password": "23User",
  "full_name": "Manager Gudang",
  "role": "manager"
}
```

### Tampilan Setelah Pengaturan Autorization dan Execute Low Stock:

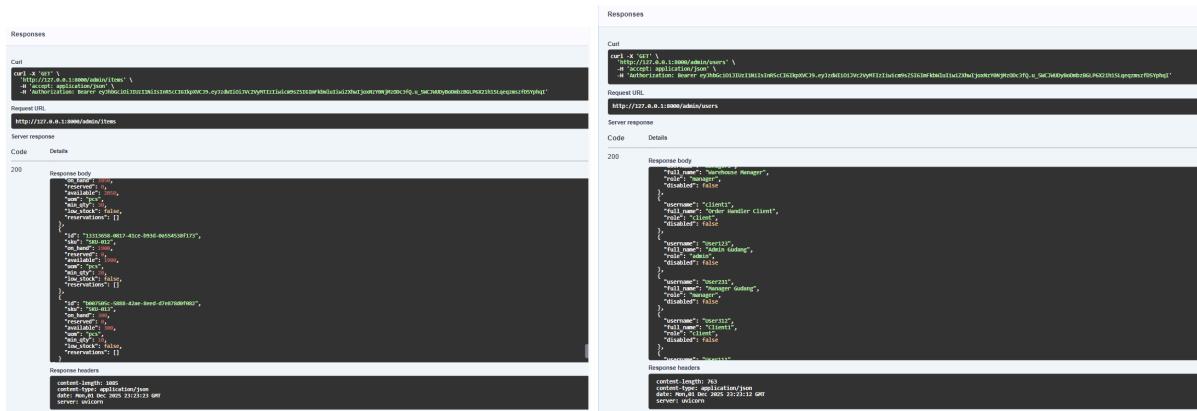


```
1 | [
2 | ]
3 | }
```

Postman juga digunakan untuk menguji error handling, seperti mencoba reservasi di luar batas stok, menghapus reservasi yang tidak ada, atau melakukan decrease stok melebihi available.



Gambar 12. Tampilan Kondisi akhir Postman Setelah Testing API di Postman



Gambar 13. Tampilan Kondisi akhir Swagger Setelah Testing API di Postman

Dari seluruh pengujian manual menggunakan Swagger UI dan Postman, seluruh endpoint Inventory Control pada *deliverable 5* berjalan sesuai dengan ekspektasi dan menghasilkan keluaran yang konsisten dengan aturan domain yang telah diimplementasikan. Setiap operasi, mulai dari pembuatan item oleh admin, proses inbound dan outbound oleh client, reservasi dan pembatalan reservasi, hingga pemantauan kondisi low stock oleh manager, telah menghasilkan respons yang tepat dan merefleksikan kondisi data yang tersimpan secara persisten di dalam database. Tidak seperti pada *deliverable 4* yang menggunakan penyimpanan *in-memory*, seluruh perubahan data pada tahap ini tercatat dalam database melalui SQLAlchemy, sehingga hasil pengujian yang dilakukan pada Swagger UI dan Postman menunjukkan konsistensi penuh antara lat dan tetap bertahan meskipun server dimulai ulang.

Selain memverifikasi perilaku domain, pengujian juga memastikan bahwa mekanisme autentikasi berbasis JWT dan kontrol akses berbasis peran berfungsi secara benar. Endpoint hanya dapat diakses ketika pengguna telah melakukan login dan membawa token yang sah, sementara percobaan akses tanpa token, menggunakan token dari role yang tidak sesuai, atau token yang telah di-logout menghasilkan penolakan sistem sebagaimana mestinya. Pada Swagger UI, mekanisme *authorization* diuji melalui fitur “Authorize” yang memungkinkan penyisipan token Bearer secara terpusat untuk seluruh request sehingga setiap endpoint dapat divalidasi aksesnya secara konsisten. Di sisi lain, Postman memfasilitasi pengujian *authorization* dengan lebih fleksibel melalui tab *Authorization* dan pengaturan Bearer Token pada setiap request, sehingga variasi skenario autentikasi dan otorisasi dapat diuji secara lebih terperinci. Hal ini menunjukkan bahwa integrasi antara lapisan autentikasi, otorisasi, API, serta *repository database* berjalan harmonis dan stabil. Secara keseluruhan, hasil pengujian pada *deliverable 5* membuktikan bahwa implementasi API, mekanisme keamanan, serta sistem penyimpanan persisten telah berfungsi dengan baik dan konsisten dengan desain arsitektur *Inventory Control* yang direncanakan pada tahap sebelumnya.

```

livenv
Ivana@LAPTOP-0GGCRACU MINGW64 ~/OneDrive/Documents/II3160_TB_18223126_WMS (main)
$ uvicorn src.main:app --reload
INFO:     Application startup complete.
INFO: 127.0.0.1:57604 - "GET / HTTP/1.1" 307 Temporary Redirect
INFO: 127.0.0.1:57604 - "GET /docs HTTP/1.1" 200 OK
INFO: 127.0.0.1:65197 - "POST /admin/items HTTP/1.1" 200 OK
INFO: 127.0.0.1:59291 - "POST /admin/items HTTP/1.1" 200 OK
INFO: 127.0.0.1:53772 - "GET /admin/items HTTP/1.1" 200 OK
INFO: 127.0.0.1:55730 - "POST /admin/items HTTP/1.1" 400 Bad Request
INFO: 127.0.0.1:52722 - "GET /admin/items/SKU-001 HTTP/1.1" 200 OK
INFO: 127.0.0.1:60875 - "POST /ohs/SKU-001/increase HTTP/1.1" 200 OK
INFO: 127.0.0.1:56282 - "POST /ohs/SKU-001/decrease HTTP/1.1" 200 OK
INFO: 127.0.0.1:59667 - "GET /ohs/SKU-001/reservations HTTP/1.1" 200 OK
INFO: 127.0.0.1:63732 - "POST /ohs/SKU-001/reserve HTTP/1.1" 200 OK
INFO: 127.0.0.1:62276 - "GET /ohs/SKU-001/reservations HTTP/1.1" 200 OK
INFO: 127.0.0.1:59039 - "POST /ohs/SKU-001/release HTTP/1.1" 200 OK
INFO: 127.0.0.1:59153 - "GET /ohs/SKU-001/reservations HTTP/1.1" 200 OK
INFO: 127.0.0.1:61215 - "GET /ohs/availability/SKU-001 HTTP/1.1" 200 OK
INFO: 127.0.0.1:59791 - "GET /manager/low-stock HTTP/1.1" 200 OK
INFO: 127.0.0.1:58305 - "POST /ohs/SKU-001/decrease HTTP/1.1" 400 Bad Request
INFO: 127.0.0.1:58147 - "POST /ohs/SKU-001/decrease HTTP/1.1" 200 OK
INFO: 127.0.0.1:61984 - "GET /manager/low-stock HTTP/1.1" 200 OK
INFO: 127.0.0.1:53482 - "POST /ohs/SKU-001/increase HTTP/1.1" 200 OK
INFO: 127.0.0.1:59225 - "GET /manager/low-stock HTTP/1.1" 200 OK
INFO: 127.0.0.1:59225 - "GET /manager/low-stock HTTP/1.1" 200 OK
INFO: 127.0.0.1:51497 - "POST /ohs/SKU-001/decrease HTTP/1.1" 400 Bad Request
INFO: 127.0.0.1:61339 - "POST /ohs/SKU-001/decrease HTTP/1.1" 200 OK
INFO: 127.0.0.1:59182 - "GET /manager/low-stock HTTP/1.1" 200 OK
INFO: 127.0.0.1:54170 - "POST /admin/items/SKU-001/threshold HTTP/1.1" 200 OK
INFO: 127.0.0.1:64495 - "GET /admin/items HTTP/1.1" 200 OK
INFO: 127.0.0.1:64440 - "GET /admin/items HTTP/1.1" 200 OK
INFO: 127.0.0.1:64586 - "POST /admin/items HTTP/1.1" 200 OK
INFO: 127.0.0.1:61248 - "GET /admin/items HTTP/1.1" 200 OK
INFO: 127.0.0.1:55866 - "GET /admin/items HTTP/1.1" 200 OK
INFO: 127.0.0.1:54150 - "GET /admin/items/SKU-003 HTTP/1.1" 200 OK
INFO: 127.0.0.1:49679 - "GET /admin/items/SKU-001 HTTP/1.1" 200 OK
INFO: 127.0.0.1:49699 - "GET /admin/items/SKU-004 HTTP/1.1" 404 Not Found
INFO: 127.0.0.1:65038 - "POST /ohs/SKU-001/increase HTTP/1.1" 200 OK
INFO: 127.0.0.1:65079 - "POST /ohs/SKU-003/decrease HTTP/1.1" 200 OK
INFO: 127.0.0.1:57299 - "POST /ohs/SKU-003/reserve HTTP/1.1" 200 OK
INFO: 127.0.0.1:57322 - "GET /ohs/SKU-003/reservations HTTP/1.1" 200 OK
INFO: 127.0.0.1:57339 - "GET /ohs/SKU-001/reservations HTTP/1.1" 200 OK
INFO: 127.0.0.1:51995 - "GET /ohs/SKU-003/reservations HTTP/1.1" 200 OK
INFO: 127.0.0.1:52005 - "POST /ohs/SKU-001/release HTTP/1.1" 400 Bad Request
INFO: 127.0.0.1:52012 - "GET /ohs/SKU-003/reservations HTTP/1.1" 200 OK
INFO: 127.0.0.1:52026 - "POST /ohs/SKU-001/release HTTP/1.1" 400 Bad Request
INFO: 127.0.0.1:52033 - "POST /ohs/SKU-003/release HTTP/1.1" 200 OK
INFO: 127.0.0.1:50833 - "GET /ohs/availability/SKU-003 HTTP/1.1" 200 OK
INFO: 127.0.0.1:50848 - "GET /manager/low-stock HTTP/1.1" 200 OK
INFO: 127.0.0.1:50687 - "GET /admin/items HTTP/1.1" 200 OK
INFO: 127.0.0.1:50730 - "GET /admin/items HTTP/1.1" 200 OK

```

Gambar 14. Tampilan Terminal VSCode saat Melakukan API Testing pada Swagger dan Postman

## C. Pengujian Menggunakan pytest

Selain pengujian manual menggunakan Swagger UI dan Postman, sistem *Inventory Control* juga diuji menggunakan pengujian otomatis berbasis pytest. Tujuan dari pengujian otomatis ini adalah memastikan bahwa logika domain maupun endpoint API berfungsi secara konsisten, dapat diuji ulang, dan tidak mudah mengalami regresi apabila dilakukan perubahan kode di masa depan. Pendekatan ini memberikan jaminan kualitas yang lebih kuat dibandingkan pengujian manual semata.

Pengujian otomatis dilakukan pada dua level utama, yaitu domain dan API. Pengujian domain berfokus pada perilaku kelas `InventoryItem` sebagai *Aggregate Root*, beserta *Value Object* dan *Entity* yang menyertainya. Skenario yang diuji meliputi penambahan stok (*increase*), pengurangan stok (*decrease*), pembuatan reservasi (*reserve*), pelepasan reservasi (*release*), serta pengecekan kondisi low stock. Semua *invariant* domain, seperti *reserved* tidak boleh melebihi `on_hand`, dijaga agar tetap konsisten. Pengujian API menggunakan `TestClient` dari FastAPI, yang memungkinkan simulasi request HTTP tanpa menjalankan server Uvicorn. Pengujian ini meliputi pembuatan item baru melalui endpoint admin, operasi *increase/decrease*, reservasi, pelepasan reservasi, serta pengecekan ketersediaan stok melalui endpoint OHS. Pendekatan ini memungkinkan pengujian *end-to-end* secara menyeluruh tanpa melibatkan infrastruktur eksternal. Seluruh pengujian dijalankan menggunakan perintah:

```

pytest -vv --color=yes --cov=src --cov-report=term-missing
# -vv : menampilkan output pytest dengan tingkat verbose tinggi, sehingga
setiap test case ditampilkan.

```

```
# --color=yes : menambahkan warna pada output untuk membedakan test yang pass dan fail.
# --cov=src : menghitung coverage pada direktori src (kode utama aplikasi).
# --cov-report=term-missing : menampilkan laporan coverage di terminal dan menandai baris kode yang belum diuji.
```

```
Ivanagl LAPTOP-OGGCRACU MINGW64 /d/IT3160_TB_IS223126_WMS (main)
$ pytest -vv --color=yes --cov=src --cov-report=term-missing
src/tests/test_domain_inventory.py::test_invariant_on_hand_negative PASSED [ 48%]
src/tests/test_domain_inventory.py::test_invariant_reserved_negative PASSED [ 50%]
src/tests/test_inventory_service.py::test_create_item PASSED [ 51%]
src/tests/test_inventory_service.py::test_create_duplicate_sku PASSED [ 53%]
src/tests/test_inventory_service.py::test_increase_stock PASSED [ 55%]
src/tests/test_inventory_service.py::test_decrease_stock PASSED [ 57%]
src/tests/test_inventory_service.py::test_decrease_stock_fail PASSED [ 59%]
src/tests/test_inventory_service.py::test_reservation PASSED [ 61%]
src/tests/test_inventory_service.py::test_reservation_not_found PASSED [ 62%]
src/tests/test_inventory_service.py::test_adjust_negative_delta PASSED [ 64%]
src/tests/test_inventory_service.py::test_invariant_reserved_exceeds_onhand PASSED [ 66%]
src/tests/test_inventory_service.py::test_is_low_stock_exact_border PASSED [ 68%]
src/tests/test_inventory_service.py::test_get_item_not_found PASSED [ 70%]
src/tests/test_inventory_service.py::test_set_threshold PASSED [ 72%]
src/tests/test_inventory_service.py::test_adjust_stock PASSED [ 74%]
src/tests/test_inventory_service.py::test_get_availability PASSED [ 75%]
src/tests/test_inventory_service.py::test_get_low_stock_items PASSED [ 77%]
src/tests/test_inventory_service.py::test_reserve_fail PASSED [ 79%]
src/tests/test_inventory_service.py::test_adjust_stock_beyond_available PASSED [ 81%]
src/tests/test_inventory_service.py::test_adjust_stock_positive PASSED [ 83%]
src/tests/test_inventory_service.py::test_get_availability_complete PASSED [ 85%]
src/tests/test_inventory_service.py::test_get_low_stock_items_empty PASSED [ 87%]
src/tests/test_inventory_service.py::test_low_stock_items_when_none_are_low PASSED [ 88%]
src/tests/test_inventory_service.py::test_low_stock_items_mixed_cases PASSED [ 90%]
src/tests/test_inventory_service.py::test_low_stock_items_large_list PASSED [ 92%]
src/tests/test_inventory_service.py::test_low_stock_items_mixed PASSED [ 94%]
src/tests/test_inventory_service.py::test_low_stock_items_many_items PASSED [ 96%]
src/tests/test_inventory_service.py::test_release_reservation_success PASSED [ 98%]
src/tests/test_inventory_service.py::test_release_reservation_not_found PASSED [100%]

=====
warnings summary =====
src/tests/test_domain_inventory.py: 10 warnings
src/tests/test_inventory_service.py: 7 warnings
<string>:6: DeprecationWarning: datetime.datetime.utcnow() is deprecated and scheduled for removal in a future version. Use timezone-aware objects to represent datetimes in UTC: datetime.datetime.now(datetime.UTC).

-- Docs: https://docs.pytest.org/en/stable/how-to/capture-warnings.html

----- coverage: platform win32, python 3.13.5-final-0 -----
Name           Stmts  Miss  Cover   Missing
-----
src\domain\inventory.py    117      0   100%
src\services\inventory_service.py    47      0   100%
-----
TOTAL          164      0   100%
===== 54 passed, 17 warnings in 0.27s =====
```

Gambar 15. Tampilan Terminal dengan Coverage Test 100%

Berdasarkan hasil eksekusi, seluruh 54 test case berhasil lulus (100% passed). Test case yang dijalankan mencakup:

- Pengujian domain: validasi SKU, penambahan/ pengurangan stok, reservasi, pelepasan reservasi, penyesuaian stok, dan pengecekan *invariant*.
- Pengujian service/API: pembuatan item, mencegah duplicate SKU, operasi *increase/decrease stock*, reservasi, *release*, *set threshold*, pengecekan *low stock*, dan pengecekan *availability*.

Selain itu, laporan coverage menunjukkan:

- src/domain/inventory.py → 100% tercakup oleh test
- src/services/inventory\_service.py → 100% tercakup oleh test
- Total coverage untuk proyek = 100%, menandakan seluruh logika bisnis penting diuji secara lengkap

Beberapa peringatan muncul terkait penggunaan `datetime.utcnow()` dan perubahan pada Pydantic, namun tidak mempengaruhi validitas hasil pengujian maupun fungsi sistem secara keseluruhan. Hasil ini membuktikan bahwa arsitektur dan implementasi modul *Inventory Control* stabil, konsisten, dan aman untuk integrasi lebih lanjut atau deployment ke lingkungan produksi. Penggunaan `pytest` dengan `coverage` memastikan seluruh alur kerja dan invariant domain tetap terjaga, sekaligus memberikan visibilitas tinggi terhadap area kode yang telah diuji.

## D. Kesimpulan Hasil Pengujian

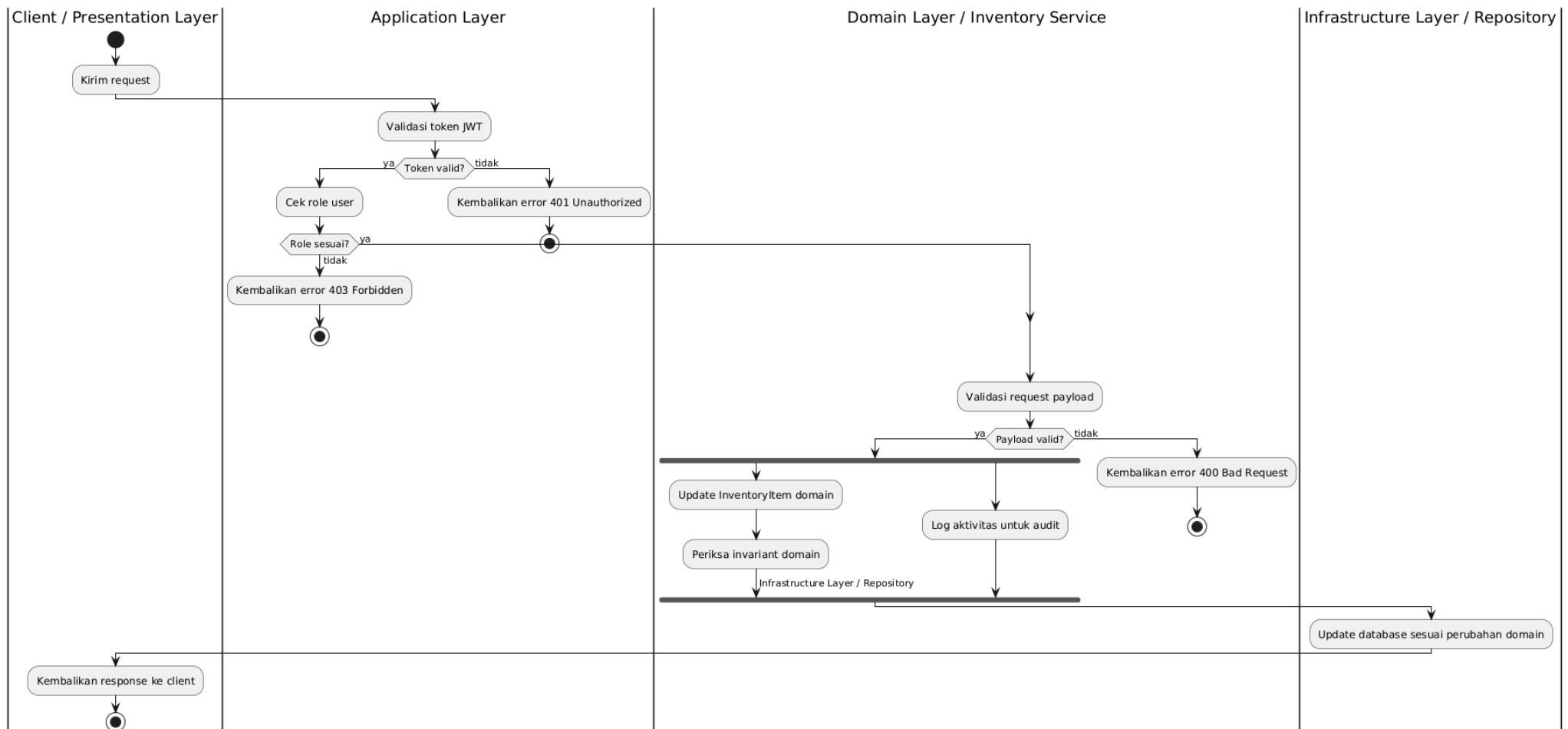
Rangkaian pengujian yang dilakukan melalui Swagger UI dan Postman, pada *deliverable 5* menunjukkan bahwa peningkatan fitur, meliputi autentikasi JWT, otorisasi berbasis peran, serta migrasi penyimpanan data ke database, berfungsi secara konsisten dan sesuai dengan desain arsitektur yang telah dirumuskan pada tahap sebelumnya. Seluruh skenario operasional yang diuji, termasuk proses *inbound* dan *outbound*, pembuatan serta pelepasan reservasi, penyesuaian stok, dan deteksi kondisi low stock, menunjukkan perilaku sistem yang sejalan dengan aturan bisnis inti yang telah ditetapkan pada tahap analisis di *deliverable 1* dan dipertegas dalam domain model pada *deliverable 3*. Hasil pengujian juga menunjukkan bahwa seluruh perubahan data yang dilakukan tercatat secara persisten dalam database, dan tetap konsisten ketika diuji melalui berbagai alat maupun setelah aplikasi dijalankan ulang.

Selain memvalidasi perilaku domain, pengujian pada *deliverable 5* juga memastikan bahwa mekanisme keamanan berjalan sesuai harapan. Pengguna hanya dapat mengakses *endpoint* berdasarkan peran yang telah ditetapkan, sementara percobaan akses yang tidak berwenang, baik karena role tidak sesuai, token tidak valid, maupun token yang telah di-logout, ditolak secara konsisten oleh sistem. Hal ini menegaskan bahwa integrasi antara autentikasi, otorisasi, API Layer, dan *Application Layer* telah diterapkan dengan benar dan stabil.

Pada *deliverable 6*, dilakukan pengujian otomatis menggunakan pytest dengan perintah `pytest -vv --color=yes --cov=src --cov-report=term-missing`. Pendekatan ini tidak hanya memverifikasi bahwa semua 54 test case berhasil dijalankan, tetapi juga menghasilkan laporan coverage kode untuk seluruh modul `src/domain/inventory.py` dan `src/services/inventory_service.py`. Hasilnya menunjukkan 100% coverage, termasuk pengujian skenario normal, batas, dan error handling, seperti penambahan dan pengurangan stok, reservasi dan pelepasan, penyesuaian stok manual, validasi low stock, serta pengecekan invariant domain. Seluruh test case berhasil lulus tanpa error, meskipun terdapat beberapa peringatan terkait penggunaan `datetime.utcnow()` dan versi Pydantic, yang tidak mempengaruhi validitas pengujian.

Secara keseluruhan, hasil pengujian dari ketiga metode ini membuktikan bahwa modul *Inventory Control* stabil, konsisten, aman, dan siap mendukung operasional sistem WMS pada tahap integrasi lebih lanjut atau deployment ke lingkungan produksi.

## ACTIVITY DIAGRAM INVENTORY CONTROL SERVICE



Gambar 16. Activity Diagram Inventory Control Service

Urutan Aktivitas pada Inventory Control Service adalah sebagai berikut.

No	Swimlane / Aktor	Aktivitas / Deskripsi	Decision / Kondisi	Output / Next Step
1	<i>Client / Presentation Layer</i>	Mengirim request ke API	–	Request diterima oleh Application Layer
2	<i>Application Layer</i>	Validasi token JWT	Token valid?	Ya → cek role user Tidak → error 401
3	<i>Application Layer</i>	Cek role user	Role sesuai?	Ya → diteruskan ke Domain Layer Tidak → error 403
4	<i>Domain Layer / Inventory Service</i>	Validasi request payload	Payload valid?	Ya → proses request Tidak → error 400
5	<i>Domain Layer / Inventory Service</i>	Proses request sesuai use case	–	Fork: jalur paralel
5a	<i>Domain Layer / Inventory Service</i>	Update <b>InventoryItem</b> domain, periksa invariant domain	–	Panggil Repository di Infrastructure Layer
5b	<i>Domain Layer / Inventory Service</i>	Log aktivitas untuk audit	–	–
6	<i>Infrastructure Layer / Repository</i>	Update database sesuai perubahan domain	–	Kembali ke Domain Layer
7	<i>Client / Presentation Layer</i>	Menerima response sesuai operasi	–	Selesai

Dari tabel di atas, terlihat alur aktivitas dari request yang dikirim client hingga response diterima kembali. Aktivitas dimulai dari validasi token dan pengecekan *role* di *Application Layer*, kemudian diteruskan ke *Inventory Service (Domain Layer)* untuk memproses request yang valid. Proses ini melibatkan dua jalur paralel: pembaruan domain **InventoryItem** dan pencatatan log untuk audit. Pembaruan domain kemudian diteruskan ke *Infrastructure Layer / Repository* untuk menyimpan perubahan ke database. Diagram ini juga menekankan penanganan error secara jelas, seperti 401 *Unauthorized*, 403 *Forbidden*, dan 400 *Bad Request*, sehingga alur normal dan alur error dapat dipahami dengan mudah.

## KESIMPULAN

Implementasi lanjutan modul Inventory Control pada *deliverable 5* berhasil memperkuat arsitektur sistem melalui penambahan mekanisme autentikasi berbasis JWT, kontrol akses berbasis peran, serta migrasi repository dari penyimpanan *in-memory* ke *database* SQLAlchemy. Struktur domain yang telah dirancang pada *deliverable 3* dan *4* tetap dipertahankan, sementara seluruh aturan bisnis inti, seperti penambahan stok, pengurangan stok, reservasi, pelepasan reservasi, dan penyesuaian stok, dijalankan dengan dukungan penyimpanan persisten. *Application Layer* tetap menjadi pusat koordinasi use case, sedangkan API FastAPI berfungsi sebagai antarmuka operasional sekaligus gerbang keamanan yang memastikan setiap endpoint hanya dapat diakses oleh aktor yang berwenang.

Pada *deliverable 6*, implementasi diperluas dengan penerapan Continuous Integration (CI) melalui GitHub Workflows dan containerisasi menggunakan Docker, sehingga proses build, pengujian, dan deployment dapat dilakukan secara otomatis dan konsisten. Infrastruktur ini memungkinkan aplikasi dijalankan baik secara lokal maupun di container, memastikan lingkungan eksekusi yang terstandarisasi dan meminimalkan risiko perbedaan konfigurasi. Selain itu, pengujian otomatis diperluas dengan pytest. Pendekatan ini tidak hanya memverifikasi bahwa seluruh test case domain dan API berhasil lulus, tetapi juga menghasilkan laporan coverage kode yang menunjukkan cakupan 100% pada modul `src/domain/inventory.py` dan `src/services/inventory_service.py`. Hasil ini mencakup pengujian skenario normal, batas, dan *error handling*, sehingga membuktikan bahwa logika domain, *Application Layer*, dan *API Layer* beroperasi konsisten, menjaga invariant, serta mempertahankan integritas data secara *end-to-end*.

Secara keseluruhan, kombinasi pengujian manual (Swagger UI dan Postman), pengujian otomatis berbasis *pytest* dengan *coverage*, serta CI dan Docker, memastikan bahwa modul *Inventory Control* telah siap untuk integrasi lebih lanjut ke dalam ekosistem WMS, dengan fondasi domain yang stabil, API yang aman, penyimpanan persisten, dan proses pengembangan yang *reproducible* serta terotomatisasi.

### Lampiran:

Link repository: [https://github.com/meerancor33/II3160\\_TB\\_18223126\\_WMS.git](https://github.com/meerancor33/II3160_TB_18223126_WMS.git)

Link video demo: <https://youtu.be/VVY45RsHAtU>